Projet Modélisation et vérification des systèmes informatiques
Projet MVSI
par Dominique Méry
1er avril 2026

Le projet MVSI reprend les notions enseignées dans le cours MVSI. L'idée est d'analyser des programmes écrits en C et d'en déduire des propriétés validées à l'aide de la plateforme TLA Toolbox. Chaque groupe prendra deux programmes et les analysera selon la méthode, en visant la correction partielle et l'absence d'erreurs à l'exécution :

— avec la méthode de traduction des algorithmes en PlusCal et la plateforme TLA Toolbox.

Chaque groupe aura deux algorithmes spécifiques. Il les traduira en un algorithme PlusCal équivalent, vérifiera la correction partielle et l'absence d'erreurs à l'exécution. Il se peut que vous trouviez une optimisation à parrtir de votre analyse et dans ce cas, vous indiquerez comment la méthode vous a permis de le faire.

Le rapport à rendre devra expliquer clairement les opérations réalisées. Ce rapport sera synthétique et mettra en avant les difficultés rencontrées et donnera des possibilités d'extension de cet outil. Le dossier final sera une archive contenant les différents éléments concernant la vérification avec chaque outil et le rapport final en pdf.

La date de remise des dossiers est le **15 mai 2026** et chaque groupe présentera son travail dans les jours suivants. Un dossier est une archive dont le nom est de la forme nom1-nom2-projet.zip (utilisez zip pour compresser) avec comme sujet "mvsi".

Les groupes seront constitués dès que possible et au plus tard le jour de la rentrée de 4 avril 2026. Les élèves enverront un fichier excel avec les différents groupes à Dominique Méry. Pour faciliter les mails et leur traitement, il est plus simple de mettre le sujet « mvsi » lors de l'envoi des courriers relatifs à ce projet.

Les sujets possibles sont donnés dans la suite et chaque groupe choisira un sujet qui devra être différent des autres groupes.

Table des matières

1 Groupe 1 **5**
1.1 GCD SCM 5
1.2 Bubble Sort 6

2 Groupe 2 **7**
2.1 Shell Sort 7
2.2 Power3 9

3 Groupe 3 **10**
3.1 Programme premier 10
3.2 Selection Sort 10

4 Groupe 4 **11**
4.1 Power5 11
4.2 Fusion de tableaux 15

5 Groupe 5 **17**
5.1 Recherche 17
5.2 Mergesort 18

6 Groupe 6 **20**
6.1 Power4 20
6.2 Bin2dec 21

7 Groupe 7 **22**
7.1 Sorting 22
7.2 Power6bis.c 26

8 Groupe 8 **27**
8.1 Quick Sort 27
8.2 Power6.c 29

9 Groupe 9 **31**
9.1 Radix Sort 31
9.2 Binary Search (iterative) 33

Listings

gcdscm.c 5
gcdscmrun.c 5
gcdscmrunrun.c 5
bubblesort.c 6
shellsort.c 7
power3.c 9
premier.c 10
selectionsort.c 10
power5.c 11
fusion.c 16
recherche.c 17
mergesort.c 18
power4.c 20
bin2dec.c 22
sorting.c 22
power6bis.c 26
quick0.c 27
power6.c 29
radixsort.c 31
binarysearchiterative.c 33

Liste des groupes avec les numéros

— Groupe 1
— Groupe 2
— Groupe 3
— Groupe 4
— Groupe 5
— Groupe 6
— Groupe 7

1 Groupe 1

1.1 GCD SCM

```
#include <limits.h>
#include  "structure.h"
#include  "gcdscm.h"

struct paire codegcdscm(int x1,int x2) {
  int y1 = x1;
  int y2 = x2;
  int y3 = x2;
  int y4 = 0;
  struct paire r;
  while (y1 != y2) {
    while (y1 > y2) { y1 = y1 -y2; y4=y4+y3;};
    while (y1 < y2) { y2 = y2 -y1;y3=y3+y4;};
  };
  r.gcd = y1;
  r.scm = y3+y4;
  r.wit= x1*x2;
   return r;
}

#include <stdio.h>
#include "structure.h"
#include "gcdscm.h"

int main () {

  int counter;
    for( counter=0; counter<5; counter++ ) {
      int x1,x2;
      struct paire r;
      printf("Enter a natural number x1:");
      scanf("%d", &x1);
      printf("Enter a natural number x2:");
      scanf("%d", &x2);
      r = codegcdscm(x1,x2);
      printf ("Result: %d,%d ----> %d,%d\n", x1,x2,r.gcd,r.scm);

    };
}

#include <stdio.h>
```

```
struct paire {
    unsigned gcd;
    unsigned scm;
  unsigned wit; };

struct paire codegcdscm(int x1,int x2) {
  int y1 = x1;
  int y2 = x2;
  int y3 = x2;
  int y4 = 0;
  struct paire r;
  while (y1 != y2) {
    while (y1 > y2) { y1 = y1 -y2; y4=y4+y3;};
    while (y1 < y2) { y2 = y2 -y1;y3=y3+y4;};
  };
  r.gcd = y1;
  r.scm = y3+y4;
  r.wit= x1*x2;
   return r;
}

int main () {

  int counter;
    for( counter=0; counter<5; counter++ ) {
      int x1,x2;
      struct paire r;
      printf("Enter a natural number x1:");
      scanf("%d", &x1);
      printf("Enter a natural number x2:");
      scanf("%d", &x2);
      r = codegcdscm(x1,x2);
      printf ("Result: %d,%d ----> %d,%d\n", x1,x2,r.gcd,r.scm);

    };
}
```

1.2 Bubble Sort

```
* Bubble sort code */

#include <stdio.h>

int main()
{
  int array[100], n, c, d, swap;

  printf("Enter number of elements\n");
  scanf("%d", &n);

  printf("Enter %d integers\n", n);

  for (c = 0; c < n; c++)
    scanf("%d", &array[c]);

  for (c = 0 ; c < ( n - 1 ); c++)
  {
    for (d = 0 ; d < n - c - 1; d++)
    {
      if (array[d] > array[d+1]) /* For decreasing order use < */
      {
        swap       = array[d];
        array[d]   = array[d+1];
        array[d+1] = swap;
      }
    }
  }

  printf("Sorted list in ascending order:\n");

  for ( c = 0 ; c < n ; c++ )
     printf("%d\n", array[c]);

  return 0;
}
```

2 Groupe 2

2.1 Shell Sort

```
 #include <stdio.h>
#include <stdlib.h>

#define SIZE 8
```

```
void display(int a[],const int size);
void shellsort(int a[], const int size);

int main()
{
    int a[SIZE] = {5,6,3,1,7,8,2,4};

    printf("--- C Shell Sort Demonstration --- \n");

    printf("Array before sorting:\n");
    display(a,SIZE);

    shellsort(a,SIZE);

    printf("Array after sorting:\n");
    display(a,SIZE);

    return 0;
}

void shellsort(int a[], const int size)
{
  int i, j, inc, tmp;

  inc = 3;
  while (inc > 0)
  {
    for (i=0; i < size; i++)
C    {
      j = i;
      tmp = a[i];
      while ((j >= inc) && (a[j-inc] > tmp))
      {
        a[j] = a[j - inc];
        j = j - inc;
      }
      a[j] = tmp;
    }
    if (inc/2 != 0)
      inc = inc/2;
    else if (inc == 1)
      inc = 0;
    else
      inc = 1;
```

```
  }
}

void display(int a[],const int size)
{
    int i;
    for(i = 0; i < size; i++)
        printf("%d ",a[i]);

    printf("\n");
}
```

2.2 Power3

```
.c#include <stdio.h>

int tp(int x)
{int  az,z,v,av,w,aw,t,at,u=0,au;
  z=0;v=0;w=1;t=3; aw=w;az=z;at=t;av=v;au=u;
  while (u< x)
        {
          printf("z=%d  ; v=%d  ; w=%d  ; t = %d; u = %d\n",z,v,w,t,u);
          au=u;
          z=az+av+aw;
          v=av+at;
          t=at+6;
          w=aw+3;
          aw=w;az=z;at=t;av=v;
          u=u+1;

        }
  return(z);
}


int main()
{
  int v;

  printf("Entrez la valeur 8 pour  v\n");
  scanf("%d", &v);
  printf(" voici la rÃ©ponse de votre solution %d --> %d\n",v,tp(v));
  return 0;
}
```

3 Groupe 3

3.1 Programme premier

```
#include<stdio.h>

int main()
{
   int n, i = 3, count, c;

   printf("Enter␣the␣number␣of␣prime␣numbers␣required\n");
   scanf("%d",&n);

   if ( n >= 1 )
   {
      printf("First␣%d␣prime␣numbers␣are␣:\n",n);
      printf("2\n");
   }

   for ( count = 2 ; count <= n ;  )
   {
      for ( c = 2 ; c <= i - 1 ; c++ )
      {
         if ( i%c == 0 )
            break;
      }
      if ( c == i )
      {
         printf("%d\n",i);
         count++;
      }
      i++;
   }

   return 0;
}
```

3.2 Selection Sort

```
#include <stdio.h>

int main()
```

```
{
   int array[100], n, c, d, position, swap;

   printf("Enter␣number␣of␣elements\n");
   scanf("%d", &n);

   printf("Enter␣%d␣integers\n", n);

   for ( c = 0 ; c < n ; c++ )
      scanf("%d", &array[c]);

   for ( c = 0 ; c < ( n - 1 ) ; c++ )
   {
      position = c;

      for ( d = c + 1 ; d < n ; d++ )
      {
         if ( array[position] > array[d] )
            position = d;
      }
      if ( position != c )
      {
         swap = array[c];
         array[c] = array[position];
         array[position] = swap;
      }
   }

   printf("Sorted␣list␣in␣ascending␣order:\n");

   for ( c = 0 ; c < n ; c++ )
      printf("%d\n", array[c]);

   return 0;
}
```

4 Groupe 4

4.1 Power5

```
#include <stdio.h>

int tp(int x)
{int a=0,b=0,c=0,d=0,e=1,f=0,g=0,h=12,i=0, j=4,k=0,l=6,m=4,p=0,
aa,ab,ac,ad,af,ae,ag,ah,ai,aj,ak,al,am;
```

```
  aa=a;ab=b;ac=c;ad=d;af=f;ae=e;ag=g;ah=h;ai=i;aj=j;ak=k;al=l;am=m;
  while (p< x)

        {
          printf("a=%d␣␣;␣b=%d␣␣;␣c=%d␣␣;␣d␣=␣%d;␣u␣=␣%d\n",a,b,c,d,e);
          aa=a;ab=b;ac=c;ad=d;af=f;ae=e;ag=g;ah=h;ai=i;aj=j;ak=k;al=l;am=m;
          a=aa+ab+ac+ad+ae;
          b=ab+af+ai+aj;
          c=ac+ak+al;
          d=ad+am;
          e=ae;
          f=af+ag+ah;
          g = ag + 24;
          h=ah;
          i=ai+12;
          j=aj;
          k = ak +12;
          l = al;
          m = am;
          p=p+1;
          aa=a;ab=b;ac=c;ad=d;af=f;ae=e;ag=g;ah=h;ai=i;aj=j;ak=k;al=l;am=m;
        }
          printf("a=%d␣␣;␣b=%d␣␣;␣c=%d␣␣;␣d␣=␣%d;␣u␣=␣%d\n",a,b,c,d,e);
  return(a);
}


int fpower5(int x)

{int a=0,b=0,c=0,d=0,e=1,f=0,g=0,h=5,i=0, j=10,k=10,l=0,m=20,o=30,p=60,z=0,aa,ab,ac

  aa=a;ab=b;ac=c;ad=d;ae=e;af=f;ag=g;ah=h;ai=i;aj=j;ak=k;al=l;am=m;ao=o;ap=p;

  while (z< x)
        {
  aa=a;ab=b;ac=c;ad=d;ae=e;af=f;ag=g;ah=h;ai=i;aj=j;ak=k;al=l;am=m;ao=o;ap=p;
          a=a+b+c+d+e;
          b=b+f+g+h;
          c=c+i +j;
          d=d+k;
          e = e + 5;
          f=f+l+m;
          g=g+o;
          i=i+o;
```

```
            h=h+20;
            j=j+30;
            k=k+20;
            l=l+p;
            m=m+60;
            o=o+60;
            p=p+120;
            z=z+1;
            //      printf("|  %d | %d | %d | %d | %d | %d | %d | %d | %d | %d |
%d\n",a,b,c,d,e,f,g,i,k,h,j,l,m,o,p,z);
   aa=a;ab=b;ac=c;ad=d;ae=e;af=f;ag=g;ah=h;ai=i;aj=j;ak=k;al=l;am=m;ao=o;ap=p;
         }
   return(a);
}


int op5(int x)

{int a=0,b=0,c=0,d=0,e=1,f=0,g=0,h=5, //i=0,
    j=10,k=10,l=0,m=20,o=30,p=60,z=0,aa,ab,ac,ad,ae,af,ag,ai,ah,aj,ak,al,am,ao,ap;

   while (z< x)
         {
            a=a+b+c+d+e;
            b=b+f+g+h;
            //      c=c+i +j;
            c=c+g+j;
            d=d+k;
            e = e + 5;
            f=f+l+m;
            g=g+o;
            //      i=i+o;
            h=h+20;
            j=j+30;
            k=k+20;
            l=l+p;
            m=m+60;
            o=o+60;
            p=p+120;
            z=z+1;
            //      printf("|  %d | %d | %d | %d | %d | %d | %d | %d | %d | %d |
%d\n",a,b,c,d,e,f,g,i,k,h,j,l,m,o,p,z);
         }
   return(a);
}
```

```
int p5(int x)

{int a=0,b=0,c=0,d=0,e=1,f=0,g=0,h=5,
    j=10,k=10,l=0,m=20,o=30,p=60,z=0,
    aa,ab,ac,ad,ae,af,ag,ah,aj,ak,al,
    am,ao,ap;

  while (z< x)
        {
          a=a+b+c+d+e;
          b=b+f+g+h;
          c=c+g+j;
          d=d+k;
          e = e + 5;
          f=f+l+m;
          g=g+o;
          h=h+20;
          j=j+30;
          k=k+20;
          l=l+p;
          m=m+60;
          o=o+60;
          p=p+120;
          z=z+1;
        }
  return(a);
}


int g(int p)

{int  az,z,aa,a,at,t,ay,y,ax,x,aw,w,av,v,au,u,i=0;
  z=0;a=12;t=6;y=4;x=0;w=1;v=0;u=0;z=0;  i=0;
  while (i< p)
        {
          z=z+u+v+w;
          v=v+t;
          t=t+12;
          w=w+4;
          u=u+x+y;
          x=x+a;
          a=a+24;
```

```
            y=y+12;
            i=i+1;
          }
  z=z*p;
  return(z);
}



int h(int p)
{int  az,z,aa,a,at,t,ay,y,ax,x,aw,w,av,v,au,u,i=0,ai;
  z=0;a=12;t=6;y=4;x=0;w=1;v=0;u=0;z=0;  i=0;ai=0;
  while (z<= p)
          {
            az= z;ai=i;
            z=z+u+v+w;
            v=v+t;
            t=t+12;
            w=w+4;
            u=u+x+y;
            x=x+a;
            a=a+24;
            y=y+12;
            i=i+1;
          }
  if (i == 0 || i==1) {return(i);}
  else {  return(ai); }

}



int main()
{
  int v;

  printf("Entrez la valeur 8 pour  v\n");
  scanf("%d", &v);
  // printf(" which is %d --> for power 4 %d et power 5
%d\n",v,g(v),fpower5(v));
  printf("%d -->  p5=%d  et fpower5  %d\n",v,p5(v),fpower5(v));
  return 0;
}
```

4.2 Fusion de tableaux

```
\begin{lstlisting}


#include <stdio.h>

void merge(int [], int, int [], int, int []);

int main() {
  int a[100], b[100], m, n, c, sorted[200];

  printf("Input␣number␣of␣elements␣in␣first␣array\n");
  scanf("%d", &m);

  printf("Input␣%d␣integers\n", m);
  for (c = 0; c < m; c++) {
    scanf("%d", &a[c]);
  }

  printf("Input␣number␣of␣elements␣in␣second␣array\n");
  scanf("%d", &n);

  printf("Input␣%d␣integers\n", n);
  for (c = 0; c < n; c++) {
    scanf("%d", &b[c]);
  }

  merge(a, m, b, n, sorted);

  printf("Sorted␣array:\n");

  for (c = 0; c < m + n; c++) {
    printf("%d\n", sorted[c]);
  }

  return 0;
}

void merge(int a[], int m, int b[], int n, int sorted[]) {
  int i, j, k;

  j = k = 0;

  for (i = 0; i < m + n;) {
```

```
    if (j < m && k < n) {
      if (a[j] < b[k]) {
        sorted[i] = a[j];
        j++;
      }
      else {
        sorted[i] = b[k];
        k++;
      }
      i++;
    }
    else if (j == m) {
      for (; i < m + n;) {
        sorted[i] = b[k];
        k++;
        i++;
      }
    }
    else {
      for (; i < m + n;) {
        sorted[i] = a[j];
        j++;
        i++;
      }
    }
  }
}
\end{lstlisting}
```

5 Groupe 5

5.1 Recherche

```
  bool jw_search ( int *list, int size, int key, int*& rec )
{
  // Basic sequential search
  bool found = false;
  int i;

  for ( i = 0; i < size; i++ ) {
    if ( key == list[i] )
      break;
  }
  if ( i < size ) {
    found = true;
```

```
    rec = &list[i];
  }

  return found;
}

  bool jw_search ( int *list, int size, int key, int*& rec )
{
  // Self-organizing (swap with previous) search
  bool found = false;
  int i;

  for ( i = 0; i < size; i++ ) {
    if ( key == list[i] )
      break;
  }
  // Was it found?
  if ( i < size ) {
    // Is it already the first?
    if ( i > 0 ) {
      int save = list[i - 1];
      list[i - 1] = list[i];
      list[i--] = save;
    }
    found = true;
    rec = &list[i];
  }

  return found;
}
```

5.2 Mergesort

```
* Helper function for finding the max of two numbers */
int max(int x, int y)
{
    if(x > y)
    {
        return x;
    }
    else
    {
        return y;
    }
}
```

```
/* left is the index of the leftmost element of the subarray; right is one
 * past the index of the rightmost element */
void merge_helper(int *input, int left, int right, int *scratch)
{
    /* base case: one element */
    if(right == left + 1)
    {
        return;
    }
    else
    {
        int i = 0;
        int length = right - left;
        int midpoint_distance = length/2;
        /* l and r are to the positions in the left and right subarrays */
        int l = left, r = left + midpoint_distance;

        /* sort each subarray */
        merge_helper(input, left, left + midpoint_distance, scratch);
        merge_helper(input, left + midpoint_distance, right, scratch);

        /* merge the arrays together using scratch for temporary storage */
        for(i = 0; i < length; i++)
        {
            /* Check to see if any elements remain in the left array; if so,
             * we check if there are any elements left in the right array; if
             * so, we compare them.  Otherwise, we know that the merge must
             * use take the element from the left array */
            if(l < left + midpoint_distance &&
                    (r == right || max(input[l], input[r]) == input[l]))
            {
                scratch[i] = input[l];
                l++;
            }
            else
            {
                scratch[i] = input[r];
                r++;
            }
        }
        /* Copy the sorted subarray back to the input */
        for(i = left; i < right; i++)
        {
            input[i] = scratch[i - left];
```

```
        }
    }
}

/* mergesort returns true on success.  Note that in C++, you could also
 * replace malloc with new and if memory allocation fails, an exception will
 * be thrown.  If we don't allocate a scratch array here, what happens?
 *
 * Elements are sorted in reverse order -- greatest to least */

int mergesort(int *input, int size)
{
    int *scratch = (int *)malloc(size * sizeof(int));
    if(scratch != NULL)
    {
        merge_helper(input, 0, size, scratch);
        free(scratch);
        return 1;
    }
    else
    {
        return 0;
    }
}
```

6 Groupe 6

6.1 Power4

```
#include <stdio.h>

int tp(int p)
{int  az,z,aa,a,at,t,ay,y,ax,x,aw,w,av,v,au,u,i=0;
  z=0;a=12;t=6;y=4;x=0;w=1;v=0;u=0;z=0;
  az=z;aa=a;at=t;ay=y;ax=x;aw=w;av=v;au=u;az=z;i=0;
  while (i< p)
        {
          printf("z=%d  ; v=%d  ; t=%d  ; w = %d; a = %d; y = %d; x = %d; u = %d;\n
          z=az+au+av+aw;
          v=av+at;
          t=at+12;
          w=aw+4;
          a=aa+24;
          y=ay+12;
          x=ax+aa;
```

```
            u=au+ax+ay;
            az=z;aa=a;at=t;ay=y;ax=x;aw=w;av=v;au=u;az=z;
            i=i+1;

          }
  return(z);
}




int tt(int p)
{int  az,z,aa,a,at,t,ay,y,ax,x,aw,w,av,v,au,u,i=0;
  z=0;a=12;t=6;y=4;x=0;w=1;v=0;u=0;z=0;

  i=0;
  while (i< p)
        {
          printf("z=%d  ; v=%d  ; t=%d  ; w = %d; a = %d; y = %d; x = %d; u = %d;\n
          z=z+u+v+w;
          v=v+t;
          t=t+12;
          w=w+4;
          u=u+x+y;
          x=x+a;
          a=a+24;
          y=y+12;
          i=i+1;

        }
  return(z);
}


int main()
{
  int v;

  printf("Entrez la valeur 8 pour  v\n");
  scanf("%d", &v);
  printf(" voici la rÃ©ponse de votre solution %d --> %d\n",v,tt(v));
  return 0;
}
```

6.2 Bin2dec

```c
#include <math.h>
#include <stdio.h>
int convert(long long n);
int main() {
    long long n;
    printf("Enter a binary number: ");
    scanf("%lld", &n);
    printf("%lld in binary = %d in decimal", n, convert(n));
    return 0;
}

int convert(long long n) {
    int dec = 0, i = 0, rem;
    while (n != 0) {
        rem = n % 10;
        n /= 10;
        dec += rem * pow(2, i);
        ++i;
    }
    return dec;
}
```

7 Groupe 7

7.1 Sorting

```c
// Bucket sort in C

#include <stdio.h>
#include <stdlib.h>

#define NARRAY 7   // Array size
#define NBUCKET 6  // Number of buckets
#define INTERVAL 10  // Each bucket capacity

struct Node {
  int data;
  struct Node *next;
};

void BucketSort(int arr[]);
struct Node *InsertionSort(struct Node *list);
void print(int arr[]);
void printBuckets(struct Node *list);
int getBucketIndex(int value);
```

```
// Sorting function
void BucketSort(int arr[]) {
  int i, j;
  struct Node **buckets;

  // Create buckets and allocate memory size
  buckets = (struct Node **)malloc(sizeof(struct Node *) * NBUCKET);

  // Initialize empty buckets
  for (i = 0; i < NBUCKET; ++i) {
    buckets[i] = NULL;
  }

  // Fill the buckets with respective elements
  for (i = 0; i < NARRAY; ++i) {
    struct Node *current;
    int pos = getBucketIndex(arr[i]);
    current = (struct Node *)malloc(sizeof(struct Node));
    current->data = arr[i];
    current->next = buckets[pos];
    buckets[pos] = current;
  }

  // Print the buckets along with their elements
  for (i = 0; i < NBUCKET; i++) {
    printf("Bucket[%d]: ", i);
    printBuckets(buckets[i]);
    printf("\n");
  }

  // Sort the elements of each bucket
  for (i = 0; i < NBUCKET; ++i) {
    buckets[i] = InsertionSort(buckets[i]);
  }

  printf("------------\n");
  printf("Bucktets after sorting\n");
  for (i = 0; i < NBUCKET; i++) {
    printf("Bucket[%d]: ", i);
    printBuckets(buckets[i]);
    printf("\n");
  }

  // Put sorted elements on arr
```

```
  for (j = 0, i = 0; i < NBUCKET; ++i) {
    struct Node *node;
    node = buckets[i];
    while (node) {
      arr[j++] = node->data;
      node = node->next;
    }
  }

  return;
}

// Function to sort the elements of each bucket
struct Node *InsertionSort(struct Node *list) {
  struct Node *k, *nodeList;
  if (list == 0 || list->next == 0) {
    return list;
  }

  nodeList = list;
  k = list->next;
  nodeList->next = 0;
  while (k != 0) {
    struct Node *ptr;
    if (nodeList->data > k->data) {
      struct Node *tmp;
      tmp = k;
      k = k->next;
      tmp->next = nodeList;
      nodeList = tmp;
      continue;
    }

    for (ptr = nodeList; ptr->next != 0; ptr = ptr->next) {
      if (ptr->next->data > k->data)
        break;
    }

    if (ptr->next != 0) {
      struct Node *tmp;
      tmp = k;
      k = k->next;
      tmp->next = ptr->next;
      ptr->next = tmp;
      continue;
```

```
    } else {
      ptr->next = k;
      k = k->next;
      ptr->next->next = 0;
      continue;
    }
  }
  return nodeList;
}

int getBucketIndex(int value) {
  return value / INTERVAL;
}

void print(int ar[]) {
  int i;
  for (i = 0; i < NARRAY; ++i) {
    printf("%d ", ar[i]);
  }
  printf("\n");
}

// Print buckets
void printBuckets(struct Node *list) {
  struct Node *cur = list;
  while (cur) {
    printf("%d ", cur->data);
    cur = cur->next;
  }
}

// Driver code
int main(void) {
  int array[NARRAY] = {42, 32, 33, 52, 37, 47, 51};

  printf("Initial array: ");
  print(array);
  printf("------------\n");

  BucketSort(array);
  printf("------------\n");
  printf("Sorted array: ");
  print(array);
  return 0;
}
```

7.2 Power6bis.c

```
#include <stdio.h>

int p3(int x)
{int  az,z,v,av,w,aw,t,at,u=0,au;
  z=0;v=0;w=1;t=3; aw=w;az=z;at=t;av=v;au=u;
  while (u< x)
        {
          printf("z=%d  ;  v=%d  ;  w=%d  ;  t  =  %d;  u  =  %d\n",z,v,w,t,u);
          au=u;
          z=az+av+aw;
          v=av+at;
          t=at+6;
          w=aw+3;
          aw=w;az=z;at=t;av=v;
          u=u+1;

        }
  return(z);
}


int p2(int x)
{int  r,k,cv,cw,or,ok,ocv,ocw;
  r=0;k=0;cv=0;cw=0;or=0;ok=k;ocv=cv;ocw=cw;
  while (k<x)
        {
          ok=k;ocv=cv;ocw=cw;
          k=ok+1;
          cv=ocv+ocw+1;
          cw=ocw+2;}
  r=cv;return(r);
}



int p23(int x)
{int r1,r;
  r1 = p2(x);
  r = p3(r1);
    return(r);}
```

```
int main()
{
  int v;

  printf("Entrez la valeur 8 pour  v\n");
  scanf("%d", &v);
  printf(" voici la rÃ©ponse de votre solution %d --> %d\n",v,p23(v));
  return 0;
}
```

8 Groupe 8

8.1 Quick Sort

```
// Quick sort in C

#include <stdio.h>

// function to swap elements
void swap(int *a, int *b) {
  int t = *a;
  *a = *b;
  *b = t;
}

// function to find the partition position
int partition(int array[], int low, int high) {

  // select the rightmost element as pivot
  int pivot = array[high];

  // pointer for greater element
  int i = (low - 1);

  // traverse each element of the array
  // compare them with the pivot
  for (int j = low; j < high; j++) {
    if (array[j] <= pivot) {

      // if element smaller than pivot is found
      // swap it with the greater element pointed by i
      i++;

      // swap element at i with element at j
      swap(&array[i], &array[j]);
```

```
    }
  }

  // swap the pivot element with the greater element at i
  swap(&array[i + 1], &array[high]);

  // return the partition point
  return (i + 1);
}

void quickSort(int array[], int low, int high) {
  if (low < high) {

    // find the pivot element such that
    // elements smaller than pivot are on left of pivot
    // elements greater than pivot are on right of pivot
    int pi = partition(array, low, high);

    // recursive call on the left of pivot
    quickSort(array, low, pi - 1);

    // recursive call on the right of pivot
    quickSort(array, pi + 1, high);
  }
}

// function to print array elements
void printArray(int array[], int size) {
  for (int i = 0; i < size; ++i) {
    printf("%d␣␣", array[i]);
  }
  printf("\n");
}

// main function
int main() {
  int data[] = {8, 7, 2, 1, 0, 9, 6};

  int n = sizeof(data) / sizeof(data[0]);

  printf("Unsorted␣Array\n");
  printArray(data, n);

  // perform quicksort on data
  quickSort(data, 0, n - 1);
```

```
  printf("Sorted array in ascending order: \n");
  printArray(data, n);
}
```

8.2 Power6.c

```
#include <stdio.h>

int p6(int x)

{int u=0,e=1,da=15,ca=20,cba=60,ba=15,bba=90,bca=60,bcba=180,aa=6,d=0,c=0,cb=0,b=0,

  while (u< x)
        {
          z=z+a+b+c+d+e;
          a=a+ad+bc+cb+aa;
          ad=b+b + bc+bc + bb+bb + ba+ba;
          b=b+bc+bb+ba;
          bc=bc+bcb+bca;
          bcb=bcb+bcba;
          bb=bb+bba;
          c=c+cb+ca;
          cb=cb+cba;

          d=d+da;
          aa = aa + 30;
          bcba = bcba + 360;
          bca = bca + 180;
          bba = bba + 180;
          ba = ba + 60;
          ca = ca + 60;
          da = da + 30;
          e = e + 6;
          cba = cba + 120;
          u=u+1;




        }
  return(z);
}
```

```
int fpower5(int x)

{int a=0,b=0,c=0,d=0,e=1,f=0,g=0,h=5,i=0, j=10,k=10,l=0,m=20,o=30,p=60,z=0,aa,ab,ac

  aa=a;ab=b;ac=c;ad=d;ae=e;af=f;ag=g;ah=h;ai=i;aj=j;ak=k;al=l;am=m;ao=o;ap=p;

  while (z< x)
        {
  aa=a;ab=b;ac=c;ad=d;ae=e;af=f;ag=g;ah=h;ai=i;aj=j;ak=k;al=l;am=m;ao=o;ap=p;
          a=a+b+c+d+e;
          b=b+f+g+h;
          c=c+i +j;
          d=d+k;
          e = e + 5;
          f=f+l+m;
          g=g+o;
          i=i+o;
          h=h+20;
          j=j+30;
          k=k+20;
          l=l+p;
          m=m+60;
          o=o+60;
          p=p+120;
          z=z+1;
          //       printf("|  %d | %d | %d | %d | %d | %d | %d | %d | %d | %d |
%d\n",a,b,c,d,e,f,g,i,k,h,j,l,m,o,p,z);
  aa=a;ab=b;ac=c;ad=d;ae=e;af=f;ag=g;ah=h;ai=i;aj=j;ak=k;al=l;am=m;ao=o;ap=p;
        }
  return(a);
}

int main()
{
  int v;

  printf("Entrez la valeur  pour  v\n");
  scanf("%d", &v);
  //  printf(" which  is  %d -->  for power 6  %d\n",v,p6(v));
  printf("%d --> p6=%d et p6*v %d\n",v,p6(v),fpower5(v)*v);
  return 0;
}
```

9 Groupe 9

9.1 Radix Sort

```
#include <stdio.h>

void printArray(int * array, int size){

  int i;
  printf("[ ");
  for (i = 0; i < size; i++)
    printf("%d ", array[i]);
  printf("]\n");
}

int findLargestNum(int * array, int size){

  int i;
  int largestNum = -1;

  for(i = 0; i < size; i++){
    if(array[i] > largestNum)
      largestNum = array[i];
  }

  return largestNum;
}

// Radix Sort
void radixSort(int * array, int size){

  printf("\n\nRunning Radix Sort on Unsorted List!\n\n");

  // Base 10 is used
  int i;
  int semiSorted[size];
  int significantDigit = 1;
  int largestNum = findLargestNum(array, size);

  // Loop until we reach the largest significant digit
  while (largestNum / significantDigit > 0){

    printf("\tSorting: %d's place ", significantDigit);
    printArray(array, size);
```

```
    int bucket[10] = { 0 };

    // Counts the number of "keys" or digits that will go into each bucket
    for (i = 0; i < size; i++)
      bucket[(array[i] / significantDigit) % 10]++;

    /**
     * Add the count of the previous buckets,
     * Acquires the indexes after the end of each bucket location in the array
                * Works similar to the count sort algorithm
     **/
    for (i = 1; i < 10; i++)
      bucket[i] += bucket[i - 1];

    // Use the bucket to fill a "semiSorted" array
    for (i = size - 1; i >= 0; i--)
      semiSorted[--bucket[(array[i] / significantDigit) % 10]] = array[i];


    for (i = 0; i < size; i++)
      array[i] = semiSorted[i];

    // Move to next significant digit
    significantDigit *= 10;

    printf("\n\tBucket: ");
    printArray(bucket, 10);
  }
}

int main(){

  printf("\n\nRunning Radix Sort Example in C!\n");
  printf("--------------------------------\n");

  int size = 12;
  int list[] = {10, 2, 303, 4021, 293, 1, 0, 429, 480, 92, 2999, 14};

  printf("\nUnsorted List: ");
  printArray(&list[0], size);

  radixSort(&list[0], size);

  printf("\nSorted List:");
  printArray(&list[0], size);
```

```
  printf("\n");

  return 0;
}
```

9.2 Binary Search (iterative)

```
#include <stdio.h>

// Iterative implementation of the binary search algorithm to return
// the position of `target` in array `nums` of size `n`

int binarySearch(int nums[], int n, int target)
{
        // search space is nums[low..high]
        int low = 0, high = n - 1;

        // loop till the search space is exhausted
        while (low <= high)
        {
                // find the mid-value in the search space and
                // compares it with the target

                int mid = (low + high)/2;       // overflow can happen
                // int mid = low + (high - low)/2;
                // int mid = high - (high - low)/2;

                // target value is found
                if (target == nums[mid]) {
                        return mid;
                }

                // if the target is less than the middle element, discard all eleme
                // in the right search space, including the middle element
                else if (target < nums[mid]) {
                        high = mid - 1;
                }

                // if the target is more than the middle element, discard all eleme
                // in the left search space, including the middle element
                else {
                        low = mid + 1;
                }
        }
```

```
        // target doesn't exist in the array
        return -1;
}

int main(void)
{
        int nums[] = { 2, 5, 6, 8, 9, 10 };
        int target = 5;

        int n = sizeof(nums)/sizeof(nums[0]);
        int index = binarySearch(nums, n, target);

        if (index != -1) {
                printf("Element␣found␣at␣index␣%d", index);
        }
        else {
                printf("Element␣not␣found␣in␣the␣array");
        }

        return 0;
}
```