Algorithmique et Structures de Données 3

SERIES of EXERCISES 1

Tutorial & Lab session: Computational complexity

10101010101010101010101010101010101010101010101010101010101010101010101010101010101010101010

EXERCISE 1: (TUTO)

Give the complexity class in Big O notation for each of the following code fragments, it is asked to do an exact calculation such as O (2N3 + 4N + 14) before deducing the answer. أحسب التعقيد الخوارزمي مع التبرير

a)
```
int sum = 0;
for (int i = 1; i <= n + 2; i++) {
    sum++;
}
for (int j = 1; j <= n * 2; j++) {
    sum += 5;}
cout << sum << endl;
```
b)
```
int sum = 0;
for (int i = 1; i <= N - 5; i++) {
    for (int j = 1; j <= N - 5; j += 2) {
        sum++;}}
cout << sum << endl;
```
c)
```
int sum = n;
for (int i = 0; i < 1000000; i++) {
    for (int j = 1; j <= i; j++) {
        sum += n;}
    for (int j = 1; j <= i; j++) {
        sum += n;}
    for (int j = 1; j <= i; j++) {
        sum += n;}}
cout << sum << endl;
```
d)
```
int sum = 0;
for (int i = 1; i <= n - 2; i++) {
    for (int j = 1; j <= i + 4; j++) {
        sum++;}
    sum++;}

for (int i = 1; i <= 100; i++) {
    sum++;}
cout << sum << endl;
```
e)
```
int sum = 0;
for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= n * n; j++) {
        sum++;
    }
    for (int j = 1; j <= 100; j++) {
        sum++;
    }
    for (int j = 1; j <= n; j++) {
        sum++;}
    sum++;}
cout << sum << endl;
```
f)
```
int sum = 0;
for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= 100; j++) {
        sum++;}}
for (int k = 1; k <= 10000; k++) {
    sum++;}
cout << sum << endl;
```
g)
```
int sum = 0;
for (int i = 0; i < n * 2; i++) {
    for (int j = 0; j < 100; j++) {
        for (int k = 0; k < j*j*j; k++) {
            sum++;}}}
cout << sum << endl;
```
h)
```
int sum = 0;
for (int i = 0; i < n * 2; i++) {
    for (int j = 0; j < i/2; j++) {
        for (int k = 0; k < n*n; k++) {
            sum++;}}}
cout << sum << endl;
```
i)
```
int count = 0;
for (int i = n; i > 0; i /= 2)
    for (int j = 0; j < i; j++)
         count++;
```

EXERCISE 2: (TUTO)

The same question of the previous exercise for the following recursive calls. نفس السؤال السابق لحالات التراجع الآتية

a)
```
int function1(int n)
{
    if (n <= 0)
        return 1;
    else
        return 1 + function1(n-1);}
```
b)
```
int function2(int n)
{   if (n <= 0)
        return 1;
    else
        return 1 + function2(n-5);}
```
c)
```
int function3(int n)
{   if (n <= 0)
        return 1;
    else
        return 1 + function3(n/5);}
```
d)

Edited by: Raouf O. Lakehal Ayat **Course coordinator: hemza.loucif@univ-msila.dz**

```
int function4(int n)
{  for(int i=0; i<n; i++)
  { function4(n-1);}
}
```

e)

```
int function5(int n)
{ for (int i = 0; i < n; i += 2) {
        // operation
  }
    if (n <= 0)
        return 1;
    else
        return 1 + function5(n-5);}
```

f)

```
void function6(int a, int b, int c)
{   if (a <= 0)
    {cout<<"func6:"<<b<<c<<endl;}
    else
    {   function6(a-1, b+1, c);
        function6(a-1, b, c+1);
    } }
```

LAB SESSION 1

C++

EXERCISE 1:

Write on your IDE the recursive functions of exercise 2 then:

- Write a main function which calls these functions
- Compile each function several times by modifying the value of the parameters and note the difference in execution time by referring to the time displayed on the console. أكتب دوال التمرين السابق حيث يتم استدعاؤها في دالة رئيسية ثم يتم تجريب قيم مختلفة وملاحظة الفروقات في وقت التنفيذ

EXERCISE 2:

You have to type on your IDE the 3 quadratic time sorting algorithms:

أكتب الدوال التالية الخاصة بخوارزميات الفرز البسيطة مع الدالة الرئيسية التي تقوم باستدعائها ونفذ كل دالة عدة مرات وفي كل مرة مع يتم وضع قيمة جديدة أكبر لـ size مع تسجيل وقت التنفيذ على ورقة في كل مرة ثم تتم مقارنة النتائج بين دالة وأخرى.

```
void bubble_sort(int *arr,int size)
{ int temp;
for(int i=size-1; i>0 ; i--)
for( int j=1;j<=i;j++)
{if(arr[j-1]>arr[j])
{ temp=arr[j-1];arr[j-1]=arr[j];
arr[j]=temp;}
}
}
void selection_sort(int *arr,int size)
{ int temp,i,min;
for(i=0; i<size; i++)
{min=i; for(int j=i+1;j<size;j++)
  if(arr[j]<arr[min]) min=j;
      if(min!=i)
{temp=arr[i];arr[i]=arr[min];arr[min]=temp;
}
}
}
void insertion_sort(int *arr,int size)
{  int current;
   for (int i=1;i<size;i++)
   {current=arr[i];int j=i;
        while(arr[j-1]>current && j>0)
            {arr[j]=arr[j-1];j--;}
     arr[j]=current;
   }
}
main()
{    int size=هنا تضع قيمة معينة وتغيرها في كل تنفيذ;
     int arr[100];
     for(int i=0;i<size;i++)
     arr[i] = rand()%1000;
   //bubble_sort(arr,size);
   //selection_sort(arr,size);
   //insertion_sort(arr,size);
   cout<<"\n sorting algorithm: "<<endl;
for(int i=0;i<size;i++) cout<<arr[i]<<" ";
}
```

rand () is a function of <cstdlib> to generate a random number, with rand()%1000 the bounds are [0.1000].

1. For the same parameter, call each function separately and note the execution time by referring to the time displayed on the console.

2. Change the parameter several times and make your conclusions.

3. Use the quicksort function below and compare the runtime results with the previous sorts.

استدع هذه المرة كود الفرز السريع التالية في دالة رئيسية وقارن زمن التنفيذ مع ما سبق

```
void swap(int &a,int &b)
{ int x=a;
      a=b;
      b=x;
}
int divide(int arr[],int left,int right)
{   int pivot=arr[left];
    int i=left;
    for(int j=left;j<=right-1;j++)
        if (arr[j]<pivot)
        {   swap(arr[i],arr[j]);
            i++;
        }
    swap(arr[i+1], arr[right]);
    return (i);
}
void quickSort(int arr[],int left,int
right)
{
    if (left<right)
    {   int p=divide(arr,left,right);
        quickSort(arr,left,p-1);
        quickSort(arr,p+1,right);
    }
}
```