<u>Lab 2</u>

Houssam Eddine ATIF

N# 610165

1/

In this procedure there is two loops, the first one run in O(n) the second one run in O(n^2) because it's nested loop.

O(n)+O(n^2) = O(n^2).

So asymptotic running time is O(n^2).

2/

A/ Algorithm Merge (Pseudo-code):

```
Algorithm merge (a, b)
  Input array a and b of n and m element. both are in order
  Output ~~merged array~~ a and b merged in array and sorted
  i = j = k ← 0;
  merged ← new array of (n + m),
  while (i + j < merged.length) do
     if i ≠ a.length && a(i) < b(j) then
          merged (k) = a(i)
          i ← i + 1
          k ← k + 1
     else if j ≠ b.length then
          merged (k) = b(j)
          j ← j + 1
          k ← k + 1
     if j = b.length then
          while i + j < merged.length do
               merged(k) = a(i)
               i ← i + 1
               k ← k + 1

  Return merged
```

B/ this algorithm compare two number from each array simultaniously in one loop (while) and put the smalest value in new array created before. So there is just one loop it means that the runing time of this algorithm is Θ(n).

C/

```java
public static int[] merge(int[] a,int[] b) {

		int i=0;
		int j=0;
		int k=0;
		int[] merged=new int[a.length+b.length];


			while(i+j<merged.length) {
				if(i!=a.length && a[i]<b[j]) {
					merged[k]=a[i];
				    i++;
					k++;
				}
				else if(j!=b.length) {
					merged[k]=b[j];
					j++;
					k++;
				}
				if(j==b.length) {
					while(i+j<merged.length) {
						merged[k]=a[i];
					    i++;
						k++;
					}
				}
			}

		return merged;

	}
```

3/ Assume the running time T(n) for a particular algorithm satisfies the following recurrence relation:

$T(1) = a$

$T(2) = b$

$T(n) = T(n-1) + T(n-1) + T(n-2) + c$ (for some a, b, c > 0)

We will use the technique of computing running time for the Fib algorithm discussed in class to solve the recurrence:

we have $T(n) = T(n-1) + T(n-1) + T(n-2) + c$

$= 2T(n-1) + T(n-2) + c$

$= 5T(n-2) + 2T(n-3) + 3c$

$\geq 5T(n-2) + 3c$

$\geq 5T(n-2)$

Lemma : Suppose $T(1) = a$, $T(2) = b$, $T(n) \geq 5T(n-2)$. We define a recurrence $S(1) = a$, $S(2) = b$, $S(n) = 5S(n-2)$. Then for all n, $T(n) \geq S(n)$

Proof: we proceed by induction on n to show $T(n) \geq S(n)$. This is obvious for n = 1 or 2. Assume $T(k) \geq S(k)$ whenever $k < n$. Then $T(n) \geq 5T(n-2) \geq 5S(n-2) = S(n)$ In particular, if it can be shown that S(n) is $\Theta(g(n))$, then T(n) is $\Omega(g(n))$.

The Guessing Method:

$S(1) = a$

$S(3) = 5*S(1) = a*a$

$S(5) = 5*S(3) = 5*5*a= 5^2c$

$S(7) = 5*S(5) = 5*5*5*c = 5^3c$

$S(9) = 5*S(7) = 5*5*5*5*c = 5^4c$

$S(n) =5^{n/2} * c$,   which is $\Theta((5)^{n/2})$

similarly, we can show S(n) is $\Theta((5)^{n/2})$ when n is even.

Claim

The function $f(n) = 5^{n/2} * c$ is a solution to the recurrence

$S(1) = a$, $S(n) = 5S(n-2)$. (when n is odd)

Proof:

Needs to show $f(1) = a$ and $f(n) = 5f(n-2)$

For n = 1, we have

$f(1) = a$

In general,

$f(n) = 5^{n/2} * c = 5*5^{(n-2)/2} * c=5f(n-2)$

as required.

By guessing method, we know that S(n) is $\Theta((5)^{n/2})$, therefore T(n) is $\Omega (((5)^{n/2})$

This shows that fib is an exponentially slow algorithm!

An algorithm is said to have an exponential running time if its running time is $\Theta(r^n)$ for some $r > 1$. We have shown here that this algorithm is either exponential or worse!

Fact: It can be shown there is a number φ for which T(n) is $\Theta(\varphi^n)$

4/

```java
public static List<List> powerSet(int[] arr) {

		int s;
		List<List> P=new ArrayList<>();
		List<List> S=new ArrayList<>();

		P.add(S);
		List<Integer> T=new ArrayList<>();


		if(arr.length==0) return P;

		for(int i=0;i<arr.length;i++){
			s=P.size();
			for(int j=0;j<s;j++) {
				T=new ArrayList(P.get(j));
				T.add(arr[i]);
				P.add(new ArrayList(T));
				T.clear();
			}
		}
		return P;
	}
```

5/

```java
public static int Fibonacci(int n) {

		int f0=0;
		int f1=1;
		int fn=f1+f0;
		int i=2;

		if(n==0) return f0;
		else if (n==1) return f1;

		while(i!=n) {
		  f0=f1;
		  f1=fn;
		  fn=f0+f1;
		  i++;
		}

		return fn;
	}
```

Running time it $\Theta(n)$ because we have just one loop.

6/

Find the asymptotic running time using the Master Formula:

$T(n) = T(n/2) + n; T(1) = 1$

=> $a=1$

$b=2$

$c=1$

$k=1$

$b^k = 2^1 = 2$

- ⇨ $b^k > a$
- ⇨ so the running time is $\Theta(n^k)$ => $\boldsymbol{\Theta(n)}$