Lab 6

Houssam Eddine ATIF

N# 610165

1/

Problem 1

We prouve in the lab 4 that if we have a binary tree of leav L and high h then $L \leq 2^h$. In this case we have 4 element need to be sorted, the number of permutation of this 4 element is $4! = 24$.

We can represent our case in a tree. the leaves of the tree is different permutations of this 4 element, and the high of this tree is the number of comparisons. so we have $24 \leq 2^h$

We need to find the smallest h that solve this equation:

$24 \leq 2^5 = 32$ because $2^4 \not\geq 24$

$\Rightarrow$ so $h = 5$ is the smallest number required of comparison in the worst case.

2/

```
public static int[] arrange(int[] a) {

		if(a.length<2) return a;

		//sorting the array
		MergeSort ms=new MergeSort();
		ms.sort(a);


		int b[]=new int[a.length];
		int j=0;

			for(int i=0;i<a.length/2;i++) {
				b[j]=a[i];
				b[j+1]=a[a.length-1-i];
				j+=2;
			}

		if(a.length%2!=0) b[a.length-1]=a[a.length/2];

		return b;
	}
```

A/ this algorithm sort an array then it arranges it, I use Merge sort algorithm (complexity O(nlog(n))) and the arrangement of this array I use one loop (complexity of this loop is O(n/2)) so this algorithm has a complexity of

$O(n\log(n))+O(n/2)= O(n\log(n))$

b/ the complexity of this algorithm depend on the sorting method used because the complexity of the sorting method is higher than O(n), and we prove in the lecture that it's impossible to obtain a comparison-based algorithm to sort an integer array, that performs better than $\Theta(n\log n)$, so this algorithm can't perform better than $\Theta(n\log n)$.

3/

We have to sort this array with the radix sort method input={80, 27, 72, 1, 27, 8, 64, 34, 16}, 9 is our radix:

Step 1:

Create a List of size 9(radix) and we place the values of the input array such as x in r[i] if x % 9 = i :

r= {27 ->72 ->27 ,1 -> 64 ,0 ,0 ,0 ,0 ,0 ,34 ->16 ,80 -> 8}

Step 2:

Create a List of size 9(radix) and we place the values of the r array such as x in q[i] if x/9 = i:

q= {1->8 ,16 ,0 ,27->27->34 ,0 ,0 ,0 ,64 ,72 ->80}

Step 3:

Scan the list q from front to back: output= {1 ,8 ,16 ,27 ,27, 34, 64, 72, 80}.

4/

```
public static int occurOnce(int[] a) {

		int[][] b=new int[3*a.length][2];
		int[][] c=new int[a.length][2];

		for(int i=0;i<a.length;i++) {
			if(b[a[i]][0]==0) {
				b[a[i]][0]=i+1;
				b[a[i]][1]++;
			}
			else {
				b[a[i]][1]++;
			}

		}

		//int j=0;
		for(int i=0;i<b.length;i++) {
			if(b[i][0]>0) {
				c[b[i][0]-1][0]=i;
				c[b[i][0]-1][1]=b[i][1];
				//j++;
			}
		}

		for(int i=0;i<c.length;i++) {
			if(c[i][1]==1) {
				return c[i][0];
			}
		}
```

```
        return 0;
    }
```

In this algorithm I have three loops the first is O(n) the second O(3n-1) an the third is O(n). so the complexity of this algorithm is O(5n-1)=O(n).