**Assignment #2 (5%)**

Submission deadline: Thursday, **February 17, 2022** **(23:59)**

**NAME: TALHA ABDULLAH PUNJABI**

**QUID: 201903446**

**Important Notes (must read):**

1. When submitting your work, you must use Blackboard, **NO other means like email submissions, are accepted.**
2. Assignments are to be solved **individually**.
3. A mark of zero (0) will be awarded for the whole assessment in which plagiarism was found to occur. Even if a single question is plagiarized**, the whole assignment will get zero** (0).
4. Submit your work as instructed below ***before*** the deadline**. No extension will be provided.**
5. Along with the MS Word submission file, you must submit separate Java files for the programs in **Questions 1 & 2**. Put all these files in a folder named **Assignment2\_QUID**. Compress this folder and submit it.
6. In the Word document, make sure that to add screenshots for input and output of your programs.

If you have any questions or doubts about any of the above-mentioned issues, please consult Eng. Alaa Hussein [alaa.hussein@qu.edu.qa](mailto:alaa.hussein@qu.edu.qa) . There are 5 questions in this assignment, each of them is 20 points. Questions 1, 2, and 4 have two sub-questions that are related to each other. Questions per sub-questions are given in the square brackets [ ] in the corresponding question.

**Q1. (Section 2.1.)**

**Q1.a.** Write a **Java program** that searches a sorted list of ***n*** items by dividing it into *three* sublists of almost ***n/3*** items. The algorithm finds the sublist that might contain the given item and divides it into *three* smaller sublists of almost equal size. The algorithm repeats this process until it finds the item or concludes that the item is not in the list. The Java program and the corresponding function(s) should be flexible enough to deal with different cases and example data should be provided in the main function of this Java program to check its correctness. **You must use the divide and conquer strategy to solve this problem.** *Hints (for example): The* ***algorithm*** *that will find the* ***location*** *of the search* ***key*** *(e.g., key) in ‘Array’ must return* ***0*** *if the* ***key*** *is not in the Array. If the* ***key*** *is in the Array, the function will return its* ***location****.*  **[15 pts]**

**Q1.b.** Analyze ***location-finding*** algorithm and give its complexity based on ***Theta*** notation [**5 pts].**

Answer: (Please write your answer here, add required space if needed)

**package** Assignment2;

**import** java.util.Scanner;

**public** **class** searchArray {

**public** **static** **void** main(String [] args) {

Scanner sc=**new** Scanner(System.***in***);

System.***out***.print("Enter the Number of items in the array :");

**int** n=sc.nextInt();

**int**[] arr=**new** **int**[n];

System.***out***.println("Enter the Sorted Array : \n");

**for**(**int** i=0; i<n;i++) {

System.***out***.print(i+"th index : ");

**int** j=sc.nextInt();

arr[i]=j;

}

System.***out***.println("Enter the term to find : ");

**int** a=sc.nextInt();

**int** z=*arraysizeCheck*(arr,a);

System.***out***.println("Returned :"+z);

}

**public** **static** **int** arraysizeCheck(**int** [] arr,**int** a) {

**int** k=0;

**int** result=0;

**if**(arr.length%3==1) {

k=1;

// for(int i=0;i<=arr.length;i++) {

Object[] arrayObjects=*dividearray*(arr,k);

**int**[] subarray1=(**int**[]) arrayObjects[0];

**int**[] subarray2=(**int**[]) arrayObjects[1];

**int**[] subarray3=(**int**[]) arrayObjects[2];

**if**(a<subarray1[0] || a>subarray3[(arr.length-k)/3]) {

System.***out***.println("Element not Found");

result=0;

}

**else** {

**if**(a<=subarray1[((arr.length-k)/3)-k])

{

result=*checkIfElementPresent*(subarray1,a)-1;

}

**else** **if**(a>subarray1[((arr.length-k)/3)-k]) {

result=*checkIfElementPresent*(subarray2,a)+((arr.length/3));

}

**else** {

result=*checkIfElementPresent*(subarray3,a)+(2\*(arr.length/3)+1);

}

}

}

**else** **if**(arr.length%3==2) {

k=2;

//for(int i=0;i<=arr.length;i++) {

Object[] arrayObjects=*dividearray*(arr,k);

**int**[] subarray1=(**int**[]) arrayObjects[0];

**int**[] subarray2=(**int**[]) arrayObjects[1];

**int**[] subarray3=(**int**[]) arrayObjects[2];

**if**(a<subarray1[0] || a>subarray3[(arr.length-k)/3]) {

System.***out***.println("Element not Found");

result=0;

}

**else** {

**if**(a<=subarray1[((arr.length-k)/3)-k])

{

result=*checkIfElementPresent*(subarray1,a)-1;

}

**else** **if**(a>subarray1[((arr.length-k)/3)-k]) {

result=*checkIfElementPresent*(subarray2,a)+((arr.length/3));

}

**else** {

result=*checkIfElementPresent*(subarray3,a)+(2\*(arr.length/3)+1);

}

}

}

**else** **if**(arr.length%3==0) {

k=0;

// for(int i=0;i<=arr.length;i++) {

Object[] arrayObjects=*dividearray*(arr,k);

**int**[] subarray1=(**int**[]) arrayObjects[0];

**int**[] subarray2=(**int**[]) arrayObjects[1];

**int**[] subarray3=(**int**[]) arrayObjects[2];

**if**(a<subarray1[0] || a>subarray3[(arr.length/3)-1]) {

System.***out***.println("Element not Found");

result=0;

}

**else** {

**if**(a<=subarray1[((arr.length-k)/3)-1])

{

result=*checkIfElementPresent*(subarray1,a)-1;

}

**else** **if**(a>subarray1[((arr.length-k)/3)-1]) {

result=*checkIfElementPresent*(subarray2,a)+((arr.length/3));

}

**else** {

result=*checkIfElementPresent*(subarray3,a)+(2\*(arr.length/3)+1);

}

}

}

**return** result;

}

**public** **static** Object[] dividearray(**int** [] arr,**int** k){

**int** n=arr.length;

**int** z=0;

**if**(k==1) {

n=(arr.length-1)/3;

z=n+1;

}

**else** **if**(k==2) {

n=(arr.length-2)/3;

z=n+2;

}

**else** {

n=arr.length/3;

z=n;

}

Object[] arrayObjects=**new** Object[3];

**int**[] arr1=**new** **int**[n];

**for**(**int** i=0;i<n;i++) {

arr1[i]=arr[i];

}

**int**[] arr2=**new** **int**[n];

**int** l=n;

**for**(**int** i=0;i<n;i++) {

arr2[i]=arr[l];

l++;

}

**int**[] arr3=**new** **int**[z];

**int** j=2\*n;

**for**(**int** i=0;i<z;i++) {

arr3[i]=arr[j];

j++;

}

arrayObjects[0]=arr1;

arrayObjects[1]=arr2;

arrayObjects[2]=arr3;

**return** arrayObjects;

}

**public** **static** **int** checkIfElementPresent(**int** [] arr,**int** a) {

**for**(**int** i=0;i<arr.length;i++) {

**if**(arr[i]==a) {

**return** i;

}

}

**return** 0;

}

}

Text

Description automatically generated

Graphical user interface, text

Description automatically generated

Text

Description automatically generated

B) Element finding algorithm has a O(logn)

As it lists and checks in the last one or the middle one or the last part of the sublist

Hence sublists divide and thus gives O(logn)

**Q2. (Section 2.1.)**

**Q2.a.** Write a **Java program** using the ***divide-and-conquer*** approach that finds the *largest* item in a list of ***n*** items. The Java program and the corresponding function(s) should be flexible enough to deal with different cases and example data should be provided in the main function of this Java program to check its correctness. **You must use the divide and conquer strategy to solve this problem, other strategies will not be graded** **[15 pts].**

**Q2.b.** Analyze this ***maximum-finding*** algorithm and present the complexity using ***Theta*** notation **[5 pts].**

Answer: (please write your answer here, add required space if needed)

**package** Assignment2;

**import** java.util.ArrayList;

**import** java.util.Scanner;

**public** **class** LargestItem {

**public** **static** **void** main(String [] args) {

Scanner sc=**new** Scanner(System.***in***);

System.***out***.println("Enter the size of the list : ");

**int** n=sc.nextInt();

System.***out***.println("Enter the list values : ");

ArrayList<Integer> list=**new** ArrayList<>();

**for**(**int** i=0;i<n;i++) {

System.***out***.print(i+"th element : ");

**int** z=sc.nextInt();

System.***out***.println("");

list.add(z);

}

**int** [] arr=**new** **int**[n];

**for**(**int** i=0;i<n;i++) {

arr[i]=list.get(i);

}

*printArray*(arr);

LargestItem item=**new** LargestItem();

item.*sort*(arr,0,arr.length-1);

*printArray*(arr);

}

**public** **static** **void** mergeArray(**int** arr[], **int** a, **int** k,**int** b)

{

**int** n=k-a+1;

**int** m=b-k;

**int**[] temp1=**new** **int**[n];

**int**[] temp2=**new** **int**[m];

**for**(**int** i=0;i<n;i++) {

temp1[i]=arr[a+i];

}

**for**(**int** g=0;g<m;g++) {

temp2[g]=arr[k+1+g];

}

**int** i=0;

**int** j=0;

**int** z=1;

**while**(i<n && j<m) {

**if**(temp1[i]<=temp2[j]) {

arr[z]=temp1[i];

i++;

}**else** {

arr[z]=temp2[j];

j++;

}

z++;

}

**while**(i<n) {

arr[z]=temp1[i];

i++;

z++;

}

**while**(j<m) {

arr[z]=temp2[j];

j++;

z++;

}

}

**public** **static** **void** sort(**int** arr[],**int** a,**int** b) {

{

**if** (a < b) {

**int** k=a+(b-a)/2;

*sort*(arr,a,k);

*sort*(arr,k+1,b);

*mergeArray*(arr,a , k ,b);

}

}

}

**public** **static** **void** printArray(**int** arr[])

{

**int** n = arr.length;

**for** (**int** i = 0; i < n; ++i)

System.***out***.print(arr[i] + " ");

System.***out***.println();

}

}

B) The average running time of this algorithm for finding the max is

Theta(nlogn)

N due to the fact that the whole line is scanned before performing the merge and then it is split down to give us the logn factor for the O(logn)

**Q3. (Section 2.3.)** Given the recurrence relation

**T (n) = 7 T (n/5) + 10n for n > 1**

**T (1) = 1**

find **T(625)**.

Answer: (please write your answer here, add required space if needed)

For the given recurrence relation:

T(625)= 7T(625/5)+10(625)= 7T(125)+6250=40551 + 6250=46801

T(125)= 7T(25)+10(125)= 7(649)+1250=5793

T(25)= 7T(5) +10(25)=7(57)+250=649

T(5)=7T(1)+10(5)=57

**Q4. (Section 2.4.)** Assume that ***Quicksort*** uses the *first* item in the list as the *pivot* item.

**Q4.a.** Give a list of ***n*** items (for example, an array of 10 integers) representing the **worst-case** scenario**. [10 pts]**

**Q.4.b.** Give a list of ***n*** items (for example, an array of 10 integers) representing the **best-case** scenario. **[10 pts]**

Answer: (please write your answer here, add required space if needed)

**Q4.a.**

Sorted ordered Numbers = [1,2,3,4,5,6,7,8,9,10]

Worst Case=

[1,2,3,4,5,6,7,8,9,10]

Since the pivot item is the first item

1,[2,3,4,5,6,7,8,9,10]

1,2,[3,4,5,6,7,8,9,10]

1,2,3,[4,5,6,7,8,9,10]

1,2,3,4,[5,6,7,8,9,10]

1,2,3,4,5,[6,7,8,9,10]

1,2,3,4,5,6,[7,8,9,10]

1,2,3,4,5,6,7,[8,9,10]

1,2,3,4,5,6,7,8,[9,10]

1,2,3,4,5,6,7,8,9,[10]

1,2,3,4,5,6,7,8,9,10

**Q4.b.**

[5,3,1,2,4,8,6,7,9,10]

Best Case

The first Item goes to the pivot item is in the middle list of items

{3,1,2,4} 5 {8,6,7,9,10}

1. {3,1,2,4} -> {1,2} 3 {4}
2. {8,6,7,9,10} -> {6,7} 8 {9,10}

1 2 3 4 5 6 7 8 9 10

**Q5. (Section 2.4.)** Use ***Quicksort*** to sort the following list. **Show the actions step-by-step following algorithm (Algorithm 2.6).**  **[20 pts]**

**123 34 189 56 150 12 9 240**

I j

123-> s= start of array

240=> e=end of array

Pivot element, start element=123

1. Search number greater than pivot element for i, than 123. Search number less than pivot element for j, than 123.

189 is greater than 123 and 9 is less than 123

Swap i and J

123 34 9 56 150 12 189 240

2) 123 34 9 56 150 12 189 240

I j

Find I and j and increment decrement. Swap i and j

123 34 9 56 12 150 189 240

3) Since i and j crossed each other, we put the pivot element in the middle of the elements. The right side is greater than 123, left side is less than 123

1. 34 9 56 123 150 189 240

4)Take Left array

12 34 9 56

Take left pivot=12

Take I and j, 12 9 34 56

Left array is sorted, 9 12 34 56

5)Take Right array

150 189 240

Right pivot=150

We reach I and j such that it is already sorted,

150 189 240

Thus,

6) Final Answer:

By using divide and conquer, we sort an array

9 12 34 56 123 150 189 240