**COLLEGE OF COMPUTING AND INFORMATICS**

**Assignment – 3**

**Course Title : Computer Programming II Course Code : CS-141**

 **Note:**

* **Submission Deadline: 18/11/2016 at 23:59**
* **5 point (50/10)**
* **Before submission, name your solution file as: *StudentId*\_CS141\_Assignment03, where *StudentId* is your student id**
* **Copying each other work will result in giving 0 to both.**
* **Will deduct 1 point for each day late.**
* **Please Note: Although the due date is after the midterm exam, but the first 3 questions deal with material included in the Midterm Exam, the last two questions will be in material included in the final**

**Question 1: [10 points]**

What is *big-Oh notation*? Complete the following table by writing the best, average, and worst complexity of the sorting algorithms

|  |  |  |  |
| --- | --- | --- | --- |
| **Sorting Algorithm** | **Best case** | **Average case** | **Worst case** |
| Selection Sort | *n2* | *n2* | *n2* |
| Insertion Sort | *n* | *n2* | *n2* |
| Merge Sort | *n log n* | *n log n* | *n log n* |
| Quick Sort | *n log n* | *n log n* | *n2* |

**Question 2: [10 points]**

Manually run the merge sort algorithm on the array {100, 10, 90, 20, 80, 30, 70, 40, 60, 50}.

**Answer:**

First sort 100, 10, 90, 20, 80,.

 Recursively, first sort 100, 10, 90.

 Recursively, first sort 100, 10.

 Recursively, first sort 100. It is sorted.

 Sort 10. It is sorted.

 Merge them: 10, 100.

 Sort 90. It is Sorted.

 Merge them: 10, 90, 100

 Recursively, sort 20, 80.

 Recursively, sort 20. It is sorted.

 Recursively, sort 80. It is sorted.

 Merge them: 20, 80.

 Merge them: 10, 20, 80, 90, 100.

 Did the same with Second sort: 30, 70, 40, 60, 50

 Merge them: 30, 40, 50, 60, 70

 Finally, merge 10, 20, 80, 90, 100 and 30, 40, 50, 60, 70 to

 10, 20, 30, 40, 50, 60, 70, 80, 90, 100

**Question 3: [10 points]**

Consider an integer array int arr[] = {100, 10, 90, 20, 80, 30, 70, 40, 60, 50} and complete the code for below insertion sort method iSort(int[] a) method to sort the array in descending order.

public void iSort(int[] a)

{

for (int i = 1; i < a.length; i++)

{

int next = a[i];

// Find the insertion location

// Move all larger elements up

int j = i;

while (j > 0 && a[j - 1] < next)

{

a[j] = a[j - 1];

j--;

}

// Insert the element

a[j] = next;

}

}

**Question 4: What is linear search discuss with Pseudo Code and Algorithm, (Or write the actual code)?** [Marks 10]

**Answer 4:** Linear search is a very simple search algorithm. In this type of search, a sequential search is made over all items one by one. Every item is checked and if a match is found then that particular item is returned, otherwise the search continues until the end of the data collection.

Pseudo code:

procedure linear\_search (list, value)

 for each item in the list

 if match item == value

 return the item's location

 end if

 end for

end procedure

Algorithm for Linear Search:

Linear Search ( Array A, Value x)

Step 1: Set i to 1

Step 2: if i > n then go to step 7

Step 3: if A[i] = x then go to step 6

Step 4: Set i to i + 1

Step 5: Go to Step 2

Step 6: Print Element x Found at index i and go to step 8

Step 7: Print element not found

Step 8: Exit

**Question 5: What is Binary Search Algorithm discuss with its complexity?**  [Marks 10]

**Answer 5:** Binary Search, also known as half-interval search is a [search algorithm](https://en.wikipedia.org/wiki/Search_algorithm) that finds the position of a target value within a [sorted array](https://en.wikipedia.org/wiki/Sorted_array)  It compares the target value to the middle element of the array; if they are unequal, the half in which the target cannot lie is eliminated and the search continues on the remaining half until it is successful.

Binary search runs in at worst [logarithmic time](https://en.wikipedia.org/wiki/Time_complexity#logarithmic_time), making [*O*](https://en.wikipedia.org/wiki/Big_O_notation)(log *n*) comparisons, where {\textstyle n}n is the number of elements in the array and *{\textstyle \log }log* is the [binary (base 2) logarithm](https://en.wikipedia.org/wiki/Binary_logarithm).

|  |  |
| --- | --- |
| **Class** | **Search algorithm** |
| Best-case performance | O(1) |
| Average performance | O(log n) |

**Algorithm**

Algorithm is quite simple. It can be done either recursively or iteratively:

1. get the middle element;
2. if the middle element equals to the searched value, the algorithm stops;
3. otherwise, two cases are possible:
	* searched value is less, than the middle element. In this case, go to the step 1 for the part of the array, before middle element.
	* searched value is greater, than the middle element. In this case, go to the step 1 for the part of the array, after middle element.

Now we should define, when iterations should stop. First case is when searched element is found. Second one is when subarray has no elements. In this case, we can conclude, that searched value doesn't present in the array.