Lab 2 - Quicksort and Merge sort

Objective: Objective of the topics is to learn about quick sort and merge sort algorithm and which one to use when. We will also learn the advantages and disadvantages of each sorting mechanism.

Quick Sort

Quick sort works in the following steps:

1. Finds a pivot (an arbitrary number)
2. Bring all numbers less than pivot to its left
3. Bring all numbers greater than pivot to its right
4. Keep doing that to left of pivot and right of pivot

The quick sort algorithm is given below:

Algorithm Quicksort(A, lo, hi):

1. **if** lo < hi **then**
2. p = **Partition**(A, lo, hi)
3. **Quicksort**(A, lo, p - 1)
4. **Quicksort**(A, p + 1, hi)

Algorithm Partition(A, lo, hi)

1. pivot = A[hi]
2. i = lo
3. **for** j = lo **to** hi - 1 **do**
4. **if** A[j] < pivot **then**
5. **swap** A[i] with A[j]
6. i = i + 1
7. **swap** A[i] with A[hi]
8. **return** i

Time complexity of Quick sort is:

- **O(nlogn)** (Average Case)
- **O(n^2)** (Worst Case)

Example: A sample input output for sorting a vector of **N** numbers using quick sort.

Sample Input	**Sample Output**
5 4 1 7 -20 41	-20 1 4 7 41

Alternate Quick Sort Algorithm:

Algorithm Quicksort(lo, hi):

```
5.  if lo < hi then
6.      p = Partition(lo, hi)
7.      Quicksort(lo, p - 1 )
8.      Quicksort(p + 1, hi)
```

Algorithm Partition(lo, hi)

```
9.  pivot = A[hi]
10. i = lo - 1
11. j = hi
12. while(true)
13.     while( A[++i] < pivot ){}
14.     while( j > 0 && A[--j] > pivot ){}
15.     if  i >= j  then
16.         break
17.     else
18.         swap A[i] with A[j]
19. swap A[i] with A[hi]
20. return i
```

Merge Sort

Merge sort works in the following steps:

1. Keep breaking down main array in half until they are just single numbers
2. Keep merging the broken parts and sort them while merging
3. Get the original array in sorted order

Time complexity of merge sort algorithm is: **O(nlogn)**

The merge sort algorithm is given below:

Algorithm Merge-Sort(A, lo, hi):

```
1.  if lo < hi then
2.      mid = (lo + hi) / 2
3.      Merge-Sort(A, lo, mid)
4.      Merge-Sort(A, mid + 1, hi)
5.      Merge(A, lo, mid, hi)
```

Algorithm Merge(A, lo, mid, hi):

```
1.  n1 = mid - lo + 1
2.  n2 = hi - mid
3.  L[1..n1] = A[lo..mid]
4.  R[1..n2] = A[mid+1..hi]
5.  L[n1 + 1] = R[n2 + 1] = INF
6.  i = j = 1
7.  for k = lo to hi do
8.      if L[i] < R[j] then
9.          A[k] = L[i]
10.         i = i + 1
11.     else
12.         A[k] = R[j]
13.         j = j + 1
```

Example: A sample input output for sorting a vector of **N** numbers in **descending** order using merge sort.

Sample Input	**Sample Output**
5 11 747 11 -7 741	747 741 11 11 -7

Tasks:

1. Write a program to sort a list of numbers in ascending order using merge sort
2. Write a program to sort a list of numbers in ascending order using quicksort.