CO322-Data Structures & Algorithms

There will be **two assignments** in total.

Each of these assignments comes in **several parts**. They will be posted one after the other. Submission deadline for each part will be given separately.

Assignment No. 1 – Part 1

All solutions must be handwritten, and a scanned copy submitted to the course page.

Copying will result in zero marks for all involved.

The marks will carry to Continuous Assessment of the course.

1.1. Sort all the functions below in increasing order of asymptotic (big-O) growth. If some have the same asymptotic growth, then be sure to indicate that. As usual, lg means base 2.

$5n$, $4 \lg n$, $4 \lg \lg n$, $n^{1/2} \lg^4 n$, 5^{5n}, $(n/4)^{n/4}$

1.2. State the ***Master theorem*** for computing complexity of recurrences and explain its meaning.

1.3. Use the ***Master theorem*** to obtain asymptotic upper and lower bounds for T(n) in each of the following recurrences:

$T(n)=4T(n/4) + 5n$; $T(n)=5T(n/4) + 4n$; $T(n) = 25T(n/5) + n^2$

1.4. Here is an implementation of **Bubble sort**:

```
{
    for (i = 1; i <= n - 1; i++)
    {
        for (j = n; j >= i + 1; j--)
            if (a[j].key < a[j - 1].key)
                swap the records a[j] and a[j - 1]
    }
}
```

a. Give the result you obtain at the end of (i-1)th iteration and indicate the progress of sorting up to then. You may assume a[0] = -∞.
b. Obtain an expression for worst-case complexity of the algorithm.

1.5. Here is an implementation of **Insertion Sort**:

```
{
    for (i = 2; i <= n; i++)
     {
      j = i ;
      while (key of a[j] < key of a[j − 1])
        {
          swap records a[j] and a[j − 1];
          j = j − 1
        }
     }
}
```

c. Give the result you obtain at the end of (i -1)th iteration and indicate the progress of sorting up to then.

d. Obtain an expression for worst-case complexity of the algorithm.

1.6. Here is a generic **Heap Sort** algorithm:

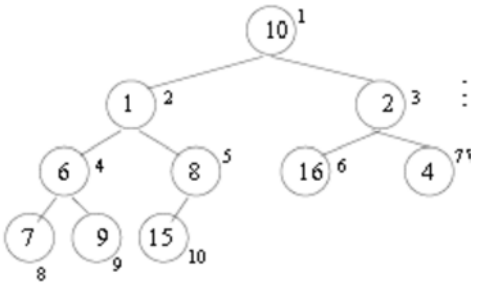
```
{
 Insert all records to form a heap S;
 while (S is not empty)
   {
    y = min (S);
    print the value of y;
    delete y from S;
   }
}
```

a. Briefly explain possible implementations of the above.

b. Heap Sort uses a function like the pushdown(first, last) which assumes that elements a[first], a[first+],…, a[last] obey the heap property, except possibly the children of a[first]. The function pushes a[first] down until the heap property is restored. Here is a possible implementation:

```
{
    for (i = n/2, i >= 1; i--)
      pushdown (i, n);   /* initial heap construction */
    for (i = n; i >= 2; i--)
     {
      swap records a[i] and a[1];
      pushdown (1, i - 1)
     }
}
```

Demonstrate the application of Heap Sort on the following input:



c. Obtain an expression for ***the worst-case time*** complexity of initial heap construction.

d. Obtain an expression for worst-case time the sorting part of the algorithm could take.

1.7. Here is an implementation of **Quick Sort**:

```
quicksort (i, j)
    {
     if (a[i] ··· a[j] contain at least two distinct keys)
      {
       let v be the larger of the first two distinct keys;
       Partition a[i] ··· a[j] so that for some k between i + 1 and j,
         a[i] ··· a[k - 1] all have keys < v, and
         a[k] ··· a[j] all have keys ≥ v;
       quicksort (i, k - 1);
       quicksort (k, j);
      }
{
```

a. Explain the role of '*v*' in the above implementation.

b. Here is an implementation of Partition function above:

```
int partition (int i; int j; key-type pivot);
      /* partitions the elements a[k] ··· a[j],
      wrt the pivot and returns the position k */
    {
      int  ℓ, r;
        {
          ℓ = i;   /* ℓ starts from left end */
          r = j;   /* r starts from right end */
          do
            swap the records a[ℓ] and a[r];
            while (a[ℓ] . key < pivot)
                ℓ = ℓ + 1;
            while (a[r] . key ≥ pivot)
                r = r − 1;
         while (ℓ ≤ r);
        return (ℓ);
        }
    }
```

Apply Quick Sort to the following input:

3	1	4	1	5	9	2	6	5	3

c. Obtain an expression for worst-case time for Quick Sort.

d. Explain what it means by sorting "*in place*".

- **End of Assignment No. 1 – Part 1** -