CS1804T – Data Structures Lab

Exercise 1 – Time Complexity Analysis

1. Consider the problem of finding k^{th} largest element from a group of N numbers. The problem may be solved in 2 ways:
   (i) Sort the numbers in descending order and retrieve the k^{th} element.
   (ii) Place the first k elements in an array and sort it in descending order. Read the remaining elements one by one and compare with k^{th} element in the array. If the element is smaller, ignore. Otherwise place the element in the correct spot in the array. When the comparisons end, the k^{th} element in the array will be the k^{th} largest element.
   a. Write 2 C programs (1 for each algorithm given above) and find the time complexity of both in Big-oh notation.
   b. Add the logic to calculate running time to your code. Calculate the actual running times of the programs for various values of n and compare results obtained.

2. A set of n integers need to be generated in a random permutation each time the code is run. Examples of legal permutations are: {4,2,1,3,5} and {2,3,1,5,4} for n=5. Each number has to occur only once and all numbers less than or equal to n should occur. You may use the *rand()* or *srand()* function for random number generation.
   Consider the following 3 algorithms:
   (i) Fill the array *a* from `a[0]` to `a[n-1]` as follows: To fill `a[i]`, generate random numbers in the range until you get one that is not already in `a[0], a[1], .. a[i-1].`
   (ii) Same as algorithm (i) but keep an extra array called *used*. When a random number is first put into the array *a*, set `used[ran]=true`. Think of how this changes the time complexity compared to algorithm (i).
   (iii) Fill the array such that `a[i] = i+1`. Then swap each number with another number picked from a random index.
   a. Write 3 C programs to implement each of the above algorithms. Give the Big-oh analysis of the expected running time of each algorithm.
   b. To each program, add the logic to calculate running time of the program. Run program (i) for n=250, 500, 1000 and 2000. Run program (ii) for n=25000, 50000, 100000, 200000, 400000 and 800000. Run program (iii) for n=100000, 200000, 400000, 800000, 1600000, 3200000, 6400000. Tabulate the actual running times in each case.
   c. Compare your analysis with actual running times.

3. What is the time complexity of the algorithm to calculate $f(x) = \sum_{i=0}^{N} a_i x^i$?
   (i) Write a program to calculate the above function. Write your own simple implementation of the exponentiation function (Don't use *pow()* function).
   (ii) Consider another algorithm for evaluation of the above function:

```
poly=0;
for(i=n; i>=0; i--)
      poly= x * poly + a[i];
```

   Show how the steps are performed by this algorithm for $x=3$, $f(x)=4x^4+8x^3+x+2$ and calculate time complexity as well as the actual running time as in the previous 2 questions.

- For each of the above questions, experiment by varying the size of data (n, k, etc.). The actual execution time of the programs will show huge differences for large values of n.
- In each question, try to find the best algorithm and understand why the algorithm is better compared to others.

Uploads in the LMS:

1. All programs to be uploaded as text.
2. Screenshots of all output.
3. Scanned copies of the written part (time complexity analysis, etc.).

You are advised to maintain a single notebook for Data Structures throughout this semester, especially for lab.