Department of Computer Science and Engineering, SVNIT, Surat

MTech I (1^{st} semester)

Design and Analysis of Algorithms

Lab Assignment No 1 - Introduction and Advanced Data Structures

AUTUMN Semester 2022-23

Dated Uploaded: 18th August 2022

Instructions:

1. *The date of submission* ***will be specified only two days before****. Therefore, right from the time the assignment is uploaded the students must start implementing the assignments.*
2. *For every delayed submission beyond the deadline, 10 marks per day will be deducted from the maximum marks of the assignment, without any exception, whatsoever may be the scapegoat.*
3. *You can use any programming language, except Python.*
4. *For every program whether specified or not, it is necessary to time your program on input dataset of a large size and give a critique of the theoretical asymptotic analysis of the algorithm that your program is based on and the empirical timing analysis of the program that you have written. The a-priori estimates and the a-posteriori analysis must be in sync with each other.*
5. *All the assignments must be submitted in the form of a zip file containing the Program Source, the screenshot of the output that you obtained, the DataSet/Input Test data used and a ReadMe .txt file explaining what platform to use, what are the input parameters required for execution and how to execute it. Also write your conclusion in ReadMe file. For the theoretical questions in Part B, write the answers using Latex in a .Tex file and submit the .tex as well as .pdf files - to be included in the zip file as above.*
6. *Perform usual error checking. Don't go overboard on this, but don't let your program die because of divide by zero.*
7. *Remember, your programs could be checked by a code-cheating program, so please follow the code of academic integrity.*
8. *There will be a viva for each assignment. This viva would be conducted for this assignment on a future date as specified.*
9. ***Maximum Points: Part A(300)***
   - *Problems at Serial nos 2, 3, 5: 5 points each (15),*
   - *Problems at Serial nos 9, 10, 18: 35 points each (105),*
   - *Rest 12 problems: 15 each (180)*
10. ***Maximum Points: Part B(100)***

P A R T - A

1. This assignment concerns creating a data set consisting of a vector of integer values. Create this data set - consisting of at least a million interger values in a vector using the functions and appropriate additional functions, as follows: Use the random number generator functions srand48(), lrand48() and/or nrand48(), to populate a vector with a million entries to serve as input data set to test your program. The vector must consist of only the positive integers. Now, write another fucntion to print the number of duplicates found in the input, if any and yet another function to round off the duplicate to the nearest *free* integral value. Edit your program to measure time required to remove the duplicates. Discuss what could be various approaches to **time** the programs, including the utilities available in Linux. Use an approach that gives you the finest resolution in time.

2. Now, assume that the new requirement is that (if not already taken care of) - your program must output the **time in microseconds**. How to achieve the same ? What approach you are able to devise as the solution to this question?

3. Now, time the function you have written using the structure *timeval* as discussed in class/-explained in the lab.

```
typedef struct timeval {
long tv_sec;
long tv_usec;
} timeval;
```

Use the program **microresolutiontimer.c** uploaded on the LMS. Compare the time obtained here with the approaches to time that you have used in the Problem #1 and 2. Give an analysis/critique of your comparison.

4. Write a program for Insertion sort, Merge sort, Quick sort and Heap sort **to count number of exchanges/swaps required** to sort the input data. Consider the worst case, the best case and the average case inputs to test your program. Use the dataset generated in the Problem #1 to serve as input data to your sorting routines to test your program.

   Now repeat the same on the sorted output. Note the difference in time - in sorting an unsorted interger vector and a sorted one. Repeat the above exercise for all the sorts. Note the time differences.

5. Do a bit of research and find out on what factors does the actual time taken by a program depends on. Use these factors as an argument to justify why it is not feasible to do empirical analysis when the goal is to compare two algorithms.

6. Write an algorithm EXPONENT(a, n) to find an using an appropriate method. Analyze the asymptotic complexity of the algorithm. Write a program to implement Binary Exponent Algorithm explained in class, time the program and analyze as explained in the instructions. Compare your implementation with the conventional method used in EXPONENT and with the *Square-and-multiply* method of exponentiation.

7. Find the cost of execution of the following code snippet by writing AN appropriate program, using the timing program and compare with the asymptotic values you obtained in the class:

   (a) Code1

```
for i = 1 to n
        for j = 1 to i
                x = x + 1
```

   (b) Code2

```
j=n
while (j >= 1){
        for i = 1 to j
                x = x + 1
        j = n/2
}
```

   (c) Code3

```
for i = 1 to n
        for j = 1 to i
                for k = 1 to i
                        x = x + 1
```

   (d) Code4

```
i=n
while (i >= 1){
for j = 1 to n
        x = x + 1
        i = i/2
}
```

8. Write a program that illustrates the comparison of an ordinary Binary tree with any one of the 2-3, AVL, or Red Black trees and illustrates the specific benefits of one of the 2-3, AVL, or Red Black trees that you have used in your implementation. Obviously for the implementation and fairer comparison, use the same data set as input. Then, extend this program to compare the 2-3, AVL, or Red Black trees with each other. Analyze the comparison and give concrete conclusions.

9. For this program, you are to fill in the functions, writing versions of (1) insertion sort (2) merge sort and (3) myownsort. myownsort() is a variant of the existing sorting algorithms that is to be devised by you with an attempt to improve upon the performance of the merge sort. Use the skeleton of the codes including here for writing your programs - filling in the functions required. The sorting functions that you write must be modeled on the illustration

of the Bubble sort shown. Assume N is the input array that you use for evaluating the programs.

After successfully writing and exeuting the code, answer the following questions:

(a) For each algorithm, how large (approximately) can N be if the sort must take no more than 10 seconds?

(b) Can you think of a quick way to improve bubble sort by a factor of 2? Write a "semibubbleSort(a)" function to do so.

Your programs must obey the following template:

```
// Program 7 —— some sorting
// compile:  gcc -o program7 program7.c
// usage:    program7 100
//    (replace "100" by the array size you want)
//     Name:
//
// Date submitted:
//
/* include the required files */
// Declarations
void bubbleSort(int a[]);
void insertSort(int a[]);
void mergeSort(int a[]);
void mySort(int a[]);
void fillRandom(int a[]);
void check(int a[]);

int N = 0;                          // Global for array size

int main(int argc, char *argv[]){
        int a[1000000];             // array to sort, N < a million
        timeval t;                          // for timing the sorts
        int starttime, endtime;     // for timing the sorts

        N = atoi(argv[1]);                  // set size of array

        // Time a bubble sort
        fillRandom(a);
        gettimeofday(&t, NULL);
        starttime = t.tv_sec;
        bubbleSort(a);
        gettimeofday(&t, NULL);
        endtime = t.tv_sec;
        printf("Bubble Sort time = %f%d%d", (endtime - starttime));
        check(a);

        // time an insertion sort
        fillRandom(a);
        gettimeofday(&t, NULL);
        starttime = t.tv_sec;
        insertSort(a);
        gettimeofday(&t, NULL);
        endtime = t.tv_sec;
        printf("Insertion Sort time = %f%d%d", (endtime - starttime));
        check(a);

        // time a merge sort
        fillRandom(a);
        gettimeofday(&t, NULL);
        starttime = t.tv_sec;
        mergeSort(a);
        gettimeofday(&t, NULL);
        endtime = t.tv_sec;
```

```
        printf("Merge Sort time = %f%d%d", (endtime - starttime));
        check(a);

        // time a "my" sort
        fillRandom(a);
        gettimeofday(&t, NULL);
        starttime = t.tv_sec;
        mySort(a);
        gettimeofday(&t, NULL);
        endtime = t.tv_sec;
        printf("MySort time = %f%d%d", (endtime - starttime));
        check(a);

        return 0;
}

void bubbleSort(int a[]){
        for(int i = 0; i <= N; i++)
        for(int j = 0; j < N-1; j++)
        if(a[j] > a[j+1]){
                int tmp = a[j];
                a[j] = a[j+1];
                a[j+1] = tmp;
        }
        return;
}

void insertSort(int a[]){

}

void mergeSort(int a[]){

}

void mySort(int a[]){

}

void fillRandom(int a[]){
        for(int i = 0; i < N; i++)
        a[i] = rand();
        return;
}

void check(int a[]){
        for(int i = 0; i < N-1; i++)
        if(a[i] > a[i+1]){
                printf("Failed to sort.  See item: %d \n\n", i) ;
                return;
        }
        printf ("Check passed \n\n");
}
```

10. Write and fill-in the statements in the sketch of the program shown below, that allows a user to interact with a heap. The heap will never have more than 20 elements in it. You should write your own additional functions to help perform these heap operations. When the heap is in "heapstate = 0," then it is just a heap, and no attempt should be made to maintain either a min or max heap property thereafter, unless a "min" or "max" command is issued. For consistency, implement the algorithms in the fashion given in the text CLRS, so that the heap will look the same for everybody after each command.

    Sample Transcript: add 1
    Added 1

1-
add 5
Added 5
1-5-
add 2
Added 2
1-5-2-
sort
Sorting
1-5-2-
max
Making a max heap
5-1-2-
sort
Sorting
1-2-5-
min
Making a min heap
1-2-5-
sort
Sorting
5-2-1-

```
// Program 8 —— some heap practice
// compile:  gcc -o program8 program8.c
// usage:    ./program8
// Name:
//
// Date submitted:
//

#include required files
#define N 20

// Global Variables
int heapstate = 0;                // 0 = heap, 1 = maxheap, 2 = minheap

void add(int A[], int heapsize, int val){
        printf("Added %d%n", val) ;
}

int extract(int A[], int &heapsize){
        printf("Extracting........%n");
}

void make_max_heap(int A[], int heapsize){
        printf("Making a max heap %n");
}

void make_min_heap(int A[], int heapsize){
        printf("Making a min heap %n");
}

void heapsort(int A[], int heapsize){
        if(heapstate == 0) return;
        printf(" %n");
}

void print(int A[], int heapsize){
        for(int i = 1; i <= heapsize; i++)
        printf("%d -%n", A[i]);
}

int main(){
```

```
    int A[N+1];
    int heapsize = 0;
    char *command = "";
    int val = 0;

    // Initialize
    // Remember that our arrays start with index 1.
    for(int i = 1; i <= N; i++)
    A[i] = 0;

    do{
            scanf("%s", command);
            if(command == "add"){
            // Add a value to the end of the heap.
            // Should maintain the heap property according to heapstate.
                    scanf(%d%, val);
                    add(A, heapsize, val);
    }

    if(command == "extract"){
            // Should extract A[1] and maintain heap property,
            // according to heapstate
            printf("Extracted %_ %n", extract(A, heapsize));
    }

    if(command == "neither"){
            heapstate = 0;
    }

    if(command == "max"){
            heapstate = 1;
            make_max_heap(A, heapsize);
    }

    if(command == "min"){
            heapstate = 2;
            make_min_heap(A, heapsize);
    }

    if(command == "sort"){
            // Sort into increasing order if heapstate = 1,
            // and into decreasing order if heapstate = 2.
            heapsort(A, heapsize);
    }

            print(A, heapsize);
    }while(command != "end");
}
```

11. You are given an undirected graph, and you should return the number of connected components in the graph. The graph will be given as a list of vertices, followed by a list of edges, as shown in the sample transcript below. Each list is terminated with the word "end."

    Your output should give the number of components, and then a sorted list of sizes of the components, from largest to smallest. Each vertex name will be a single character; in fact, a lower-case letter. Please make sure that your output matches the sample transcript shown below, for the ease of the grading. You do not need to do error checking on the input.

    Sample Transcript:c
    $ program9
    Please input your vertices:
    a
    b

12. A **concatenate** operation takes two sets, such that all the keys in one set are smaller as compared to all the keys in the other set, and merges them together. Design and implement

an algorithm to concatenate two binary search trees into one binary search tree. Analyze the worst case time of your algorithm and time the program you have implemented for the same algorithm. This should be $O(h)$ where h is the maximal height of the two trees. Implement and do the empirical analysis of your algorithm.

13. Design a data structure to maintain a set of elements, each with a key and a value. The following operations should be supported:
    - *Find_value(x)*: find the value associated with the element x (is nil of x is not in the set)
    - *Insert(x,y)*
    - *Delete(x).*
    - *Add(x,y):* add the value y to the current value of the element with key x.
    - *Add_all(y):* add the value y to the values of all the elements in the set.

    The worst case running time should be $O(logn)$ for each of these operations. Implement and do the empirical analysis of your algorithm.

14. Design a data structure to support the following operations:
    - *insert(x, T)* - Insert item x into the set T.
    - *delete(k, T)* - Delete the k^{th} smallest element from T.
    - *member(x, T)* - Return true if x ϵ T.

    All operations must take $O(logn)$ time on an n-element set.

15. Write a function (and a driver program to run it) to compare whether two binary trees are identical. Identical trees have the same key value at each position and the same structure. Implement and do the empirical analysis of your algorithm.

16. Sachin Tendulkar has been asked by the coach Rahul Dravid to complete a job. The job is of partitioning $2n$ indian cricket team/pool players into two teams/pools of n players each. Each player has a numerical rating that measures how good he/she is at the game. He seeks to divide the players as unfairly as possible, so as to create the biggest possible talent imbalance between team A and team B. Show how the Sachin can do the job in $O(nlogn)$ time. Implement and do the empirical analysis of your algorithm.

17. Let S be a sorted array of n integers. Give an algorithm that finds the pair $x, y \epsilon S$ that minimizes $|x - y|$, for $x \neq y$. The algorithm must run in O(n) worst-case time. Implement and do the empirical analysis of your algorithm.

18. We have looked at the theoretical analysis of algorithms using asymptotic notations. In practice, however, a theoretically inferior algorithm (I) can work better than a theoretically superior algorithm (S) due to the constants hidden in the big-Oh analysis. Also, several optimizations that can be done for a simpler algorithm may not quite be applicable for S. However, in all cases, S should eventually outperform I. Finally, whether inferior or superior, blackboard algorithms do not consider all possible situations, something which a programmer has to deal with.
    So, what is the Task at hand in this assignment ?

    Well, in this programming assignment, you will solve the problem of multiplying two polynomials (not necessarily of the same degree) using two different methods.

    (a) The first method, call it simple, is the straightforward quadratic method.
    (b) The second method, call it $nlog3$, is a divide-and-conquer (D&C) method which runs in $\theta(nlog3)$. You will subdivide the input polygons based on even or odd indices rather than the "first half" and "second half method". Clearly you must first figure out the algorithm

    Input Format Please read this section carefully and make sure that you understand it clearly. You will read the input from three input files. The first two files contain two sets of polynomials to be multiplied. You may assume that they are called $first.in$ and *second.in*.

    Each of the two files contains a set of coefficients that describes a set of polynomials as follows.

    - The coefficients are listed in ascending order of power.
    - Each line represents exactly one real number coefficient written in ASCII.

- The number of coefficients is not expected to be small. The precision is expected to be up to three decimal points. For example, if first.in is in the following format, then this means that it describes the set of polynomials $15, 15 + 0.5x$, and $15 + 0.5x + x^2$.
  15
  0.5
  1
- The third input file will list a set of indices of coefficients (starting from zero for the polynomial of degree zero) that needs to be output each on a separate line. You may assume that the coefficient\s file is called coeff.in. For example, the following file lists the indices 0, 2, and 1.
  0
  2
  1
  Your task is best illustrated by an example as follows. Assume first.in and coeff.in are as before, and second.in is as follows.
  1
  1
  That is second.in describes the set of polynomials 1 and $1 + x$.

Your task will be to read the input files one line from each of the three files at a time, compose the two polynomials up to that line, multiply the two polynomials using the two methods, output the running time of the two methods (in milliseconds), and output the value of the corresponding coefficient which is listed on the same line of coeff.in (the exact output format will be described in Output Format Section).
For example, for the previous input files you will do the following.

(a) Read the polynomial 15 of degree 0 from first.in, the polynomial 1 of degree 0 from second.in, and the coefficient index 0 from coeff.in. Multiply the two polynomials 15 and 1 using the two methods. Output the running time for the two methods. Output the value of the coefficient index 0 of the resultant polynomial which in this case equals to 15

(b) Read the polynomial 15+0.5x of degree 1 from first.in, the polynomial 1 + x of degree 1 from second.in, and the coefficient index 2 from coeff.in. Multiply the two polynomials 15 + 0.5x and 1 + x using the two methods. Output the running time for the two methods. Output the value of the coefficient index 2 of the resultant polynomial which in this case equals to 0.5.

(c) Read the polynomial 15 + 0.5x + x2 from first.in, the polynomial 1 + x + 0x2 from second.in, and the coefficient index 1 from coeff.in. Notice that second.in has already reached the end of file marker in the previous step, so you have to handle the case of polynomials of different lengths as well as polynomials of equal lengths. Also, notice that coeff.in will always have number of lines equal to the number of lines in the longer file of first.in and second.in (the file for the longer polynomial). Multiply the two polynomials 15+0.5x+x2 and 1+x+0x2 using the two methods. Output the running time for the two methods. Output the value of the coefficient index 1 of the resultant polynomial which in this case equals to 15:5.

(d) Now first.in, second.in, and coeff.in have reached the end of file marker, and your task is now complete.

Output Format Please read this section carefully since any deviation in the output format will result in loss of points. Your output should consist of two parts as we discussed in previous section. The corresponding values of the coefficients listed in coeff.in, AND the running time of the two methods, both as text files. You may assume that the output files are called coeff.out and running.out respectively. The running.out file should consist of three columns text file. The first column represents n, which is the degree of the multiplied polynomials (the longer one), the running time for simple in milliseconds, and the running time for nlog3 in milliseconds. This file is intended to be used to draw a plot in order to show the asymptotic performance of both algorithms for each value of n and then see if it actually matches the theoretical results.
For example, running.out can be as follows.
0 1000 800
1 2000 1700
.
.

.
100 50000 43700
The coeff.out file should be the actual answers to the input given above. Specifically, It should list the values of the required coefficients based on the coeff.in input file as we described in Section 4. You should output each of the coefficients as a real number on a separate line with a precision up to three decimal points. For example, the output for the example we discussed in previous section should be in the following format.
15
0.5
15.5

PART - B

1. Write the answers to the following questions and submit the .tex and the .pdf files in the .zip file as explained in the instructions, with the same assignment.Yes, your answers must be written using Latex and hence your .zip file must contain the .tex source, too.

   (a) IF $f(n) = 100 * 2^n + 8n^2$, prove that $f(n) = O(2^n)$. Can you claim that $f(n) = \theta(2^n)$. IF so, prove the same.

   (b) Is it correct to say that $f(n) = 3n + 8 = \Omega(1)$?. Given the facts that f(n)=$3n+3$=$\Omega(n)$ and f(n)=$3n+3 = \Omega(1)$, which one is correct ? Which one would you choose to prescribe the growth rate of f(n) ?

   (c) Consider the two functions viz. $f(n) = n^2$ & $g(n) = 2n^2$. Which functions growth rate is higher ? USe appropriate asymptotic notation to specify the time complexity of the two functions.

   (d) Prove the following: *For any two functions f(n) and g(n),* $f(n) = \theta(g(n))$ *only if f(n)=O(g(n) and f(n)=Ω(g(n)).*

   (e) Solve the following problems:

      i. Show that $T(n) = 1 + 2 + 3 + ...n = \Theta(n^2)$

      ii. Prove or disprove: $2n^3 – n^2 = O(n^3)$

      iii. Prove that $7n^2logn + 25000n = O(n^2logn)$

   (f) If $T1(n) = O(f(n))$ and $T2(n) = O(g(n))$ then show that (a) $T1(n) + T2(n) = max(O(g(n), O(f(n))$ (b) $T1(n) * T2(n) = O((g(n) * (f(n))$

   (g) Show that $max\{f(n), g(n)\} = \Theta(f(n) + g(n))$

   (h) Prove or disprove: (a) $n^2 2^n + n^{100} = \theta(n^2 2^n)$ (b) $n^2/logn = \Theta(n^2)$

   (i) Prove that if T(x) is a polynomial of degree n, then $T(x) = \Theta(x^n)$.

   (j) If P(n) is any polynomial of degree m or less then show that $P(n) = a^0 + a_1 n + a_2 n^2 + \ldots\ldots\ldots\ldots\ldots + a_m n^m$ then $P(n) = O(n^m)$

   (k) Find the running time of the following algorithm in terms of the asymptotic notations:

```
Algorithm SUM(n)
1.        answer = 0;
2.        for i= 1 to n do
3.            for j= 1 to i do
4.                  for k = 1 to j do
5.                    answer++;
6.        print(answer);
```

   (l) Let A and B be two programs that perform the same task. Let $t_{A(n)}$ and $t_{B(n)}$ respectively denote their values. For each of the following pairs, find the range of n value for which program A is faster than program B :

      i. $t_{A(n)} = 1000n$ and $t_{B(n)} = 10n^2$

      ii. $t_{A(n)} = 1000nlog_2 n$ and $t_{B(n)} = n^2$

      iii. $t_{A(n)} = 2n^2$ and $t_{B(n)} = n^3$

      iv. $t_{A(n)} = 2n$ and $t_{B(n)} = 100n$

   (m) Consider an input array A of n elements. Each element is an n-bit integer except 0. Which sorting algorithm would you recommend for sorting the array ? Why ? What will be the complexity your sorting algorithm ? [Hint: What is the range in which each array value (i.e. a number) i.e. an integer falls into ?]

   (n) Given the following statement viz. *Consider an input array $a[1..n]$ of arbitrary numbers. It is given that the array has only $O(1)$ distinct elements. What does the statement imply ?*