CS2351 Data Structures

Homework 2

Due (Part 1): 11:59 pm, March 23, 2010

Due (Parts 2 & 3): 11:59 pm, March 30, 2010

In this assignment, you may choose to work alone or work in pairs, though the minimum requirements will be different. If you work in pairs, please submit your file together as one copy, and mark clearly your names and your university numbers in the submitted file.

Part I. Written (50%)

1. (15%) Assume $T(1) = 1$. Solve the following recurrences:

   (a) $T(n) = T(n/5) + T(3n/4) + n$. *Hint:* Use substitution or recursion tree method.

   (b) $T(n) = 8T(n/2) + n^3$. *Hint:* Use Master Theorem.

   (c) $T(n) = 8T(\sqrt{n}) + (\log n)^3$. *Hint:* Changing variable.

2. Quick Sort is a very practical algorithm that can sort an array $A[1..n]$ of n distinct numbers. It works recursively as follows:

```
QuickSort(Array $A$, Length $n$)
{
    if ($n \leq 1$)
        return $A$;

    Pick an arbitrary element $x$ from $A$;
    Partition the other elements of $A$ into 2 groups, $A_{small}$ and $A_{large}$, such that
        $A_{small}$ = all elements with value smaller than $x$ ;
        $A_{large}$ = all elements with value larger than $x$ ;

    Use QuickSort to sort $A_{small}$ ;
    Use QuickSort to sort $A_{large}$ ;
    return sorted $A_{small}$, followed by $x$, followed by sorted $A_{large}$;
}
```

   (5%) Show that QuickSort is correct.

   (5%) Show that in the worst case, the running time of QuickSort is $\Omega(n^2)$.

   (5%) The above algorithm assumes that all the numbers in A are distinct. What will happen if they may be non-distinct?

   (5%) Briefly explain how to modify the above algorithm so that it can handle the case where numbers may not be distinct.

3. Your friend, John, gives you another array $B[1..n]$ of numbers. He wants to obtain the smallest k numbers in B, in sorted order.

   One way to solve this is: Sort B and the output the k smallest numbers. This can be done in $O(n \log n)$ time, using the sorting algorithms we have learnt before.

(15%) Design an $O(n+k\log n)$-time algorithm to help John find the desired k numbers, and explain why it runs in $O(n+k\log n)$ time. *Hint:* Use heap.

Part II. Programming (35%)

In this part, you are asked to implement the Heap Sort algorithm, and the Quick Sort algorithm. If you work alone, you can just implement Heap Sort. If you work in pairs, you will need to implement both algorithms.

You can develop your code using either C or C++. Your code should be compilable using Dev-C++. We will assess your code by the correctness (25%) and the readability (10%). Note that you should not use any predefined system calls or standard template library calls to perform the Heap Sort or the Quick Sort.

Specifications:

Your program of both implementations will expect two arguments when it runs. The first one gives the filename of the input file that contains the number to be sorted, and the second one gives the filename of the output file where the sorted numbers are stored. For example, suppose your program name is `heapsort`, the input filename is `input.txt`, and the output filename is `output.txt`. Then when we run your program, we will type: `./heapsort input.txt output.txt` so that your program sorts the numbers in the `input.txt` and store the result in `output.txt`.

The format of the input file is as follows: The first line stores an integer n that indicates how many numbers are to be sorted. The second line stores n integers which are the numbers to be sorted. For example, the input file may look like:

```
10
14  36  5  2  8  100  13  -12  4  -13
```

The format of the output file is simply the sequence of the sorted numbers, each separated by a single space. For example, the output file of the previous example should look like:

```
-13  -12  2  4  5  8  13  14  36  100
```

Remarks:

When you are implementing a heap of at most n numbers, you will just need to use an array A of size n, and use a counter k to remember the current size of the heap. At any point, the k numbers in the heap will be stored in the array entries $A[1..k]$, as discussed in the lecture notes. Insertion will then correspond to adding a number at $A[k+1]$, updating k, and then performing the necessary swapping (up the tree). Extract-Min will then correspond to putting the number $A[k]$ to $A[1]$, updating k, and then performing the necessary swapping (down the tree).

To help the implementation, you may declare some easy helper functions, such as:

```
// Assume that array A[1..n] represents the heap, and
// there is a global variable k storing the current size of A
// Given the index x of a node, return the index of its parent in $A$
int parent(int $x$)
{ return $x$/2 ; }

// Given the index $x$ of a node, return the index of its left child
// Return -1 if left child does not exist
int left(int $x$)
{
    if ( 2 * $x$ > k )
        return -1 ;
    else
        return 2 * $x$ ;
}
```

With the help of the above functions (and other similar functions), we can perform Insert or Extract-Min easily.

Part III. Experiments (15%)

In this part, you are asked to test the running time of several sorting algorithms on your machine. If you work alone, you can compare Heap Sort and the sorting algorithms you have completed in Homework 1.If you work in pairs, you will need to compare Quick Sort in addition.

To get a better understanding of which algorithm is the best on your machine, we shall test the sorting algorithms on input array of various sizes (such as 10, 100, 1000, 10000, ...). For each size n, we shall generate a few random arrays of size n, and get the average time of running the sorting algorithm on these arrays. Include the following details in your submitted file:

- The configuration of your machine (such as machine model, CPU speed, RAM, OS, ...).
- Indicate which algorithms you have tested, and how you implement them. Also indicate which compiler you use (and the compilation options if any).
- Indicate the number of random arrays where you take the average timing.
- Explain how you generate the random arrays.
- Plot your timing results in a graph, and estimate roughly for which input size which algorithm is the best.

The purpose of the above is to allow people, who are interested in your work, to repeat your experiments and obtain the same results.