Indian Statistical Institute

Semester-I 2017-2018

M.Tech.(CS) - First Year

Lab Test IV (2 December, 2017)

Subject: Data and File Structures Laboratory

Total: 60 marks + 15 bonus marks Duration: 4 hrs.

SUBMISSION INSTRUCTIONS

1. Naming convention for your programs: `cs17xx-test4-progy.c`
2. When you have finished, copy all your files to `~dfslab/2017/labtest4/cs17xx/`.

If your program does not take input in the specified format, your code will not be evaluated, and you will get no credit.

1. *Missing parentheses* (20 marks).

   Suppose you are given an arithmetic expression from which all left parentheses have been erased. You have to write a program that reads such an expression from stdin, and prints to stdout, the equivalent expression with the missing parentheses inserted. For example, given the input:

   $$1 + 2) * 3 - 4) * 5 - 6)))$$

   your program should print

   $$((1 + 2) * ((3 - 4) * (5 - 6)))$$

   You may assume that the arithmetic expression involves only single digit integers, and the standard binary arithmetic operators $(+, -, *, /)$.

   HINT: Use a stack S. Read the input from left to right, one symbol (digit, operator or right parenthesis) at a time. If the symbol just read is a digit or operator (e.g., 1, + or 2 in the sample test case given above), push it onto S. If the symbol is a right parenthesis, pop an operand (2), an operator (+) and yet another operand (1). Push a left parenthesis, followed by the operands and operator just popped (in proper order), and the right parenthesis. Note that an operand may itself be an expression. For instance, in the example given above, when the penultimate right parenthesis is encountered, the operands and operators that need to be popped are $(5 - 6)$, $*$, and $(3 - 4)$.

   Source: Algorithms 4e, Sedgewick and Wayne.

2. *Bag packing* (20 marks).

   You are given n books, each having an integer weight between 200 gm and 1000 gm. You have an endless supply of very large plastic bags, each of which can carry a total weight of 1000 gm. Write a program that uses the following heuristic algorithm to compute the number of bags required to pack all n books.

Consider the n books one by one in the order in which they are given to you. For each book, look at the bags in *increasing* order of remaining capacity, and put the book into the first bag that can carry the weight of the book.

For example, if you are given 4 books weighing 300 gms, 800 gms, 550 gms, and 200 gms, you should pick the first book (300 gms), select a new bag (B_1), and put the first book into it. Next, look at the second book. It is too heavy to fit in B_1, so you should again select a new bag (B_2), and insert the book into the second bag. Now consider the bags in *increasing* order of remaining capacity: B_2 (remaining capacity: 200 gms), B_1 (remaining capacity: 700 gms), B_3 (new bag, remaining capacity: 1000 gms), B_4 (as for B_3), B_5, ... The first bag that the third book can be put into is B_1. After this is done, the new ordering of bags is: $B_1, B_2, B_3, B_4, \ldots$. The first bag that the last book can be put into is B_2. Thus, the total number of bags required is 2.

Input / output convention: the weights of the books will be specified as command-line arguments; your program should print the number of bags required as shown in the examples below.

```
$ ./cs17xx-test4-prog2 300 800 550 200
2
```

NOTE: For full credit, your program should take no more than $O(n \log n)$ time. You may use a search tree (preferably balanced) to store the remaining capacities of the bags that are currently in use. Given a book weighing w gms, you should use the `ceiling` function to identify the bag in which the book should be inserted (recall that the `ceiling` function returns the smallest element in the tree that is greater than or equal to the given value). Also note that the remaining capacity of the selected bag will have to be updated after the book is inserted into it.

3. You are given two unsorted arrays A and B containing m and n integers. We define an AB-sum as any sum of the form $A[i] + B[j]$ where $1 \leq i \leq m$ and $1 \leq j \leq n$ (note that we are using one-based indexing for this question). Assume that A and B are such that the mn AB-sums are all distinct. Given a positive integer k, your task is to write a program to find out the k smallest AB-sums. For each of the algorithms given below, write a program to implement the algorithm.

   (a) **Algorithm A** (10 marks). Store the mn AB-sums in an array S, sort S, and report the k smallest elements from the sorted S. This requires $O(mn \log(mn))$ time and $O(mn)$ extra space.

   (b) **Algorithm B** (10 marks). Convert S to a min-heap (in place). For k times, print the minimum followed by deleting the minimum. This improves the running time to $O(mn + k \log(mn))$. However, the extra space requirement remains the same.

   (c) **Algorithm C** (15 marks, **bonus**). If we assume $k \leq m/\log m$ and $k \leq n/\log n$, we can design an algorithm which runs in $O(m + n)$ time and uses only $O(k)$ extra space. We first convert A and B to two min-heaps. Assume that indexing in the arrays A and B is one-based, and $A[i]$ and $B[j]$ refer to the indices i and j **after** these arrays have been converted to min-heaps.

   We use a priority queue Q for storing pairs (i, j) of indices satisfying $1 \leq i \leq m$ and $1 \leq j \leq n$. Q is heap ordered with respect to the sum $A[i] + B[j]$. Given i and j, we can easily compute $A[i] + B[j]$, so this sum need not be explicitly stored in Q. Since A and B are min-heaps, $A[1]$ and $B[1]$ are the smallest elements in A and B. Thus, $A[1] + B[1]$ is the smallest AB-sum. So

we first insert $(1, 1)$ into Q. Then, we enter a loop which runs until k sums are printed. Let (i, j) be the minimum (with respect to $A[i] + B[j]$) stored in Q. We print $A[i] + B[j]$, and delete (i, j) from Q. We then insert the four index pairs $(2i, j), (2i + 1, j), (i, 2j)$, and $(i, 2j + 1)$ in Q (check, before insertion, that $2i, 2i + 1 \leq m$ and $2j, 2j + 1 \leq n$).

Input / output convention: m, followed by the m elements of A, n, followed by the n element of B, k. All input will be provided via stdin; the output (k smallest AB-sums) should be printed to stdout. See the example below.

```
$ ./cs17xx-test4-prog3A
64          <-------- m (elements of A below)
647 225 200 137 823 241 923 173 633 797 203 797 820 789 338 72 274 407 577
306 167 928 40 417 86 751 384 697 144 986 665 468 225 121 372 143 86 737 86
4 557 874 341 628 148 748 852 212 50 656 681 153 353 824 8 176 783 993 559
970 936 399 61 882
60          <-------- n (elements of B below)
206 354 757 547 700 623 14 623 514 646 194 444 414 849 125 566 202 948 292
96 732 285 374 702 940 772 762 737 974 559 620 898 631 96 445 331 437 177
672 951 822 866 395 955 715 520 240 636 187 532 731 637 535 823 339 475 314
819 931 7
10          <-------- k
11 15 18 22 47 54 57 64 68 75   Output (k smallest AB-sums)
```

Source: `http://cse.iitkgp.ac.in/~abhij/course/lab/Algo1/Autumn16/A4.pdf`