Ann, Bob, Charlie (*replace with your names*)
COSC 336
3/19/2020

Assignment 5

Instructions.

1. Due October 14.

2. This is a team assignment. Work in teams of 3-4 students. Submit one assignment per team, with the names of all students making the team.

3. Your programs must be written in Java.

4. Write your programs neatly - imagine yourself grading your program and see if it is easy to read and understand. At the very beginning present your algorithm in plain English or in pseudo-code (or both). Comment your programs reasonably: there is no need to comment lines like "i++" but do include brief comments describing the main purpose of a specific block of lines.

5. You will submit on Blackboard 2 files. The first file should be a .pdf file with the solution to Exercise 1and with descriptions in English or in pseudocode of the algorithms for the programming task you are required to do and the results that you are required to report. Make sure you label the results as indicated below. Also insert images/screenshots with the output you obtain for each testing data. The second file will contain the Java sources of your two programs.

   For editing the above document with Latex, see the template posted on the course website.

   assignment-template.tex and

   assignment-template.pdf

   To append in the latex file a pdf file, place it in the same folder and then include them in the latex file with

   ```
   \includepdf[pages=-,pagecommand={},width=\textwidth]{file.pdf}
   ```

   To append in the latex file a .jpg file (for a photo), use

   ```
   \includegraphics[width=\linewidth]{file.jpg}
   ```

Exercise 1 (has two questions:)

1. **Exercise 6.1-6, textbook page 154.**

2. **Exercise 6.1-7, textbook, page 154.**

   Hint: You need to show that in the tree view of a heap the nodes indexed with $\lfloor n/2 \rfloor + 1, \lfloor n/2 \rfloor + 2, \ldots, n$ do not have children (so they are leaves) and also that the nodes indexed with $1, 2, \ldots, \lfloor n/2 \rfloor$ have at least one child (and consequently they are not leaves). You can consider separately the case when $n = 2k$ (n is even), and $n = 2k + 1$ (n is odd). See also the formulas on page 152.

Programming Task 1.

The input consists of a sequence of numbers $a[1], a[2,] \ldots, a[n]$. Your task is to design an $O(n^2)$ algorithm that finds an increasing subsequence with the maximum possible sum. An increasing subsequence is given by a sequence of indices $1 \leq i_1 < i_2 < \ldots < i_k \leq n$ such that $a[i_1] \leq a_{[}i_2] \leq \ldots \leq a_{[}i_k]$. Note the the indices defining the subsequence are not necessarily consecutive numbers. The program will output the max sum and the increasing subsequence with that sum.

Your algorithm should work in time $O(n^2)$.

For example, for sequence $1, 14, 5, 6, 2, 3$, the increasing subsequence $1, 14$ has the largest sum $1 + 14 = 15$. ($1, 5, 6$ is another increasing subsequence but its sum, $1 + 5 + 6 = 12$ is smaller.)

Input specification: the first line contains n and the second line contains $a_1, \ldots, a_n$. Numbers on the same line are separated by spaces. You may assume that n is not bigger than $10,000$ and all the numbers fit in int.

Output specification: the output contains the maximum possible sum of an increasing subsequence (note: you do not have to compute the subsequence itself, just the maximum possible sum).

Sample inputs :
input-5.1.txt
input-5.2.txt
Sample outputs :
answer-5.1.txt
answer-5.2.txt

Test your program on the following inputs:
input-5.3.txt
input-5.4.txt
and report in a table the results you have obtained for these two inputs. Label the results with appropriate text, for example "output for file input-*.*.txt", or something similar.

Hint: You should use a dynamic programming algorithm. For each $i \leq n$, define
$s[i]$ = max sum of an increasing subsequence with last element $a[i]$, and
$p[i]$ = index of the element preceding $a[i]$ in an increasing subsequence with max sum and last element $a[i]$.

You start with $s[1] = a[1]$ and $p[1] = -1$ (-1 is a just a conventional value that indicates that $a[1]$ has no predecessor), and then in order you compute $s[2], s[3], \ldots s[n]$ and $p[2], \ldots, p[n]$. You need to think how to compute $s[i]$, using the preceding $s[j]$, $j < i$. At the end, you get the the subsequence in reverse order from last element to the first element. More precisely, first you find the last element of an increasing subsequence with max sum, and next, starting from the last element, you go from predecessor to predecessor and construct the subsequence.