Lab 7 - Priority Queues

1. Carry out the array-based version of HeapSort on the input array
   [1, 4, 3, 9, 12, 2, 4]

   A. For this part, do Phase I and Phase II of sorting as described in class – in Phase I, show the steps for populating a heap from the input sequence, then show the steps of Phase II as shown in class

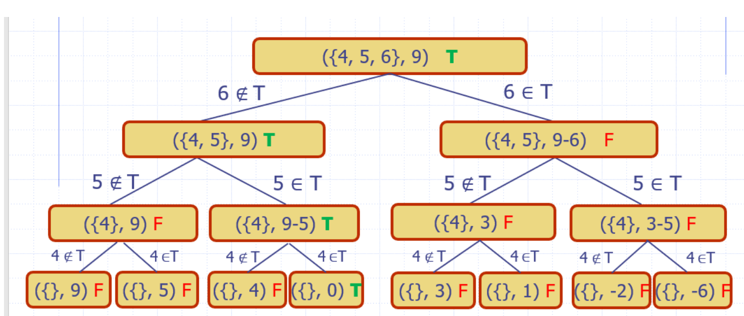
   B. For this part, do Phase I using BottomUpHeap and then complete the sorting with Phase II. Is the heap that you get in A the same as the one you obtain in B with BottomUpHeap?

2. *SubsetSum Again.* Recall that the SubsetSum problem is the following: You are given a set S = {s_0, s_1, . . ., s_{n-1}} of positive integers and a non-negative integer k. Is there a subset T of S whose sum is k?

   Earlier in the course, we gave a recursive algorithm for solving the SubsetSum problem, shown below. A demo in class showed that this recursive algorithm (like the fib(n) algorithm) has many redundant computations.

Algorithm *RecSubsetSum*(*S*, len, *k*)
   Input: $S = \{s_0, s_1, \ldots, s_{n-1}\}$ positive integers,
      k nonnegative integer, len = S.length
   Output: *true* if for some $T \subseteq S$ we have
      sum(*T*) = *k*, else *false*
   //base case
   if *S.size*() = 0 **then**
      if *k* = 0 **then return** *true*
      else return *false*
   [lastIndex, last] ← *S.removeLast*()
   result1 ← *RecSubsetSum*(*S*, lastIndex, *k*)
   if *result1* **then**
      return *result1*
   result2 ← *RecSubsetSum*(*S*, lastIndex, *k* – *last*)
   return *result2*

   A. Show that RecSubsetSum has a worst-case running time of $\Omega(2^n)$. *Hint.* Try counting self-calls. Do this by examining the recursion tree for the algorithm running on a set S of size n (a typical example is shown below). Count the number of nodes in this tree.



B. Improve RecSusbsetSum by storing computations and re-using them when needed. Rewrite the pseudo-code shown above so that these storing/re-using steps are included.

C. What is the running time of your improved recursive algorithm for SubsetSum?

3. *Backpack Problem.* The BackPack Problem is a relative of the SubsetSum problem and an one uses the same approach to solve it as we used to solve SubsetSum.

> *The Backpack Problem.* You are given an array S = $\{s_0, s_1, . . ., s_{n-1}\}$ of items, together with a corresponding arrays weights[] = $\{w_0, w_1, \ldots, w_{n-1}\}$, representing the weights of the items, and values[] = $\{v_0, v_1, \ldots, v_{n-1}\}$, representing the values of the items. You are also given constants V (for "min value") and W (for "max weight"). Does there exist a subset T of S for which the sum of values is $\geq$ V and the sum of weights is $\leq$ W?

The idea is that you are trying to get the as many of the most valuable items from S into your backpack as possible, without exceeding the weight limit W. If the sum of the values of the items you pick out is $\geq$ V, then the answer to the problem is "true".

A. Describe a brute-force solution to the Backpack Problem and write it up in pseudo-code.

B. Let $\mathcal{A}$ be any algorithm that solves the Backback Problem (this means that it outputs "true" when it should and it outputs "false" when it should). $\mathcal{A}$ could be the algorithm you described in part A, but it doesn't have to be that one. Show how to use $\mathcal{A}$ as the essential part of an algorithm that solves the SubsetSum problem.

C. Give a recursive algorithm to solve the Backpack Problem. Use the recursive solution to SubsetSum as a model. *Hint.* Try proving the following about the Backpack Problem first:

Lemma. Let S, weights[], values[], W, V be inputs for the Backback Problem. Then

(a) If there is a subset T' of the set S' containing the first $n - 1$ elements of S (namely s_0, s_1, …, s_{n-2}) so that either of the following occurs, then there is a subset T of S that solves the original problem

(i) The sum of the weights of items in T' is $\leq$ W and sum of the values of items in T' is $\geq$ V

(ii) The sum of the weights of items in T' is $\leq$ W-w_{n-1} and the sum of the values of items in T' is $\geq$ V-v_{n-1}

(b) If it is possible to find a subset T' of S' for which either (i) or (ii) holds, then there is a subset T of S that solves the original problem.

D. What is the running time of your recursive algorithm?