COMP 3711 Design and Analysis of Algorithms
2015 Fall
Solutions to Assignment 2

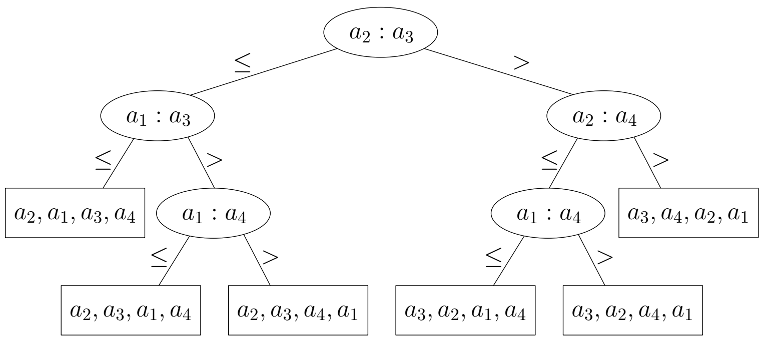
1. (a) $O(n \log n)$

   (b) Build the heap: $O(n)$. Call Extract-Min k times: $O(k \log n)$. Total time: $O(n + k \log n)$.

   (c) Build the heap with repeated insertions: $O(n \log n)$. Call Extract-Min k times: $O(k \log n)$. Total time: $O(n \log n)$.

   (d) We first use the linear-time selection algorithm to find the k-th smallest number (denoted as a_k), which takes $O(n)$ time. Then find all the k numbers smaller than or equal to a_k, which takes $O(n)$ time. Finally, we sort these k numbers, which takes $O(k \log k)$ time. So the total running time is $O(n + k \log k)$.

2. The subtree T_3:



3. We use divide-and-conquer combined with counting sort. The first call to the following recursive algorithm is SortString$(A, 1, n, 1)$.

```
SortString(A, p, r, i):
if p ≤ r then return;
Sort A[p..r] by the i-th character using counting sort;
foreach group of strings with the same i-th character do
    Suppose the current group is A[j..k];
    d ← j − 1;
    for t ← j to k do
        if A[t] has only i characters then
            d ← d + 1;
            Swap A[t] and A[d];
    // This puts all strings with only i characters at the beginning of the group;
    SortString(A, d + 1, k, i + 1);
```

Analysis: We can't use standard recursion analysis since we don't know the the size of each sub-problem. However, we observe that each call to SortString(A, p, r) takes time linear in the array size $p - r + 1$, not counting the recursive calls. So we just need to count, for each string s in A, how many SortString calls involve it. We see that a string s with i characters will only be involved in i SortString calls, so the total running time is $\sum_s(\text{length}(s)) = O(n)$.

4. The algorithm: Put the first base station at $x + 4$ where x is the coordinate of the first house. Remove all the houses that are covered and then repeat if there are still houses not covered.

   Correctness: Let X be the solution returned by this greedy algorithm, and let Y be an optimal solution. Consider the first base station where Y is different from X. Suppose the base station in X is located at x and the one in Y is located at y. By the greedy choice, we must have $x > y$. Now move y to x in Y . The resulting Y must still cover all houses. Repeatedly applying this transformation will convert Y into X. Thus X is also an optimal solution.