# Assignment 2: CSE548: Analysis of Algorithms

Shivasagar Boraiah — #112077826

October 2, 2018

**Problem 1.** Textbook [Kleinberg & Tardos] Chapter 4, page 190, problem #7.

*Proof.* ..
Input: n jobs, to be processed on one supercomputer and then on n set of PCs. Job Ji takes pi time on Supercomputer and fi time on PC.
Output: Polynomial-time greedy algorithm that finds a solution with as small a completion time as possible.
As we have n PCs and one Supercomputer and as two jobs are independent of each other, ordering the jobs according to their execution time on supercomputer has no effect on the optimal algorithm. To make use of all the PCs we should order the jobs such that all n jobs will be kept for processing parallel in PC and they finish in same time. So, we should arrange the jobs according to finishing time of PC.
Let N = {J1, J2.....Jn-1, Jn} be the scheduler and let us arrange them in the increasing order of increasing finishing time fi. By doing this, Job Ji finishes step one at pi and Job Jj will enter step one where as job Ji will enter PC (step two). In the worst case, chances are more that by the time Jj finishes step one, Ji would have finished processing in PC. This is not an optimal solution because job Jj will again get into same PC keeping other PCs idle and increasing the total completion time.
Let us arrange F = {J1, J2, ..... Jn-1, Jn} in a decreasing order of PC time. Now, when job, Ji finishes Pi it will enter PC1 with its finishing time of fi which is greater than any other jobs. Then, Job Jj finishes step one with Pj and enters PC2 in parallel with Ji to finish within fj. This way all PCs will be kept in run and minimizes the total completion time for n jobs. Clearly F is the optimal solution or at least a better solution. set N can be made equal to F by interchanging the adjacent jobs in the decreasing order of fi.

```
Job(F):
  arrange the Jobs in decreasing order of fi;
i=0;
while i < n:
  Ji enters supercomputer with finishing time pi
  for j in PC(n):
  Ji runs in PC(j) in a finishing time of fi;
```

□

**Problem 2.** Textbook [Kleinberg & Tardos] Chapter 4, page 190, problem #13.

*Proof.* ...
Input: n jobs, ti total time of a ith job, Ci completion time of a job i, wi weight associated with each job.
Output: An algorithm to minimize the weighted sum of the completion times.
Idea: If we order the jobs in decreasing order of wi/ti and schedule them, it will be our optimal solution. We can prove the optimality of this algorithm using "exchange argument" thought in class (Student homework scheduling).
sort the jobs by its weight over time take to complete.
S = {w1/t1 > w2/t2 > ..> wi/ti >... > wn/tn }
Let us consider i, j in S and j,i in S' that are in order with wj/tj >= wi/tj. If we are able to prove that swapping j, i in S' as in S will not increase the cost, then we have arrived at an optimal solution.
The goal is to take the non-optimal solution and remove all the inverted pairs to arrive at a optimal solution. So, lets consider S' and try to swap i and j. Swapping these two will effect the overall completion time of an algorithm. With swapping the overall time C is definitely lesser or equal to C' the time without swap. So, we can continue this way and swap all such elements in S' so that every time we either reduce or make it equal to the overall completion time.

Job(S):
  Order the jobs in the decreasing order of wi/ti;

□

**Problem 3.** Textbook [Kleinberg & Tardos] Chapter 4, page 190, problem #17.

*Proof.* ...
Input: n requests to schedule an Interval that runs 24-hours a day
Output: Set of optimal requests that can be kept run all day.
Idea: We have discussed Interval Scheduling problem in class just that in this problem the set of requests will not end, they continue circularly 24-hours everyday. As the requests will start at midnight and can end in daytime and vice verse, let us pick up one request, i that will start at midnight and lets remove all other requests that is conflicting with the ith request. Now, select that request j that has less no. of conflicts. With this, the rest of the problem can be solved using the interval scheduling solution. Starting with picking one midnight request, we will arrive at an optimal solution with the minimal clash that run circularly everyday. This algorithm will run in O($n^2$) time.

Request(N):
  Order the requests with maximum time occupied in midnight
  Iterate through the n requests taking one midnight request at a time
  Arrive at the optimal list of requests with less number of conflicts.
  run the requests circularly everyday.

□

**Problem 4.** Textbook [Kleinberg & Tardos] Chapter 4, page 190, problem #20.

*Proof.* ...
Input: Graph G=(V, E) with altitude, ae associated with it.

Output: To prove MST of G is also a Minimum Altitude Graph and vice verse.
Idea: (a) By definition, let T be a MST of G and let A be a Minimum altitude graph. So, lets use contradiction property as in class to prove this wrong. If T and A are different then there musty two different paths, PT and PA, such that PT has minimum path for u-v edge that belongs to ae where as PA has minimum height. Consider the edges PT, PA - uv for which u'v' in PA. With this we can construct a path along PT to u' and to u and from u to v to v' and finally to PA, which is possible. This violates cycle property as graph, G U u'v' is a cycle. Hence, this is a contradiction and T is also a minimum altitude tree.
(b) Let A = (V, E') be a minimum altitude sub-graph that does not contain edge e of graph G. Using Cut property, removing e from G results in 2 sub-graphs resulting in partitions G and G-A. edge e with one vertices in G and another in G-A. Also, to find a shortest route, Graph G has to reach from u to v and it cannot use edge, e which contradicts that there is another higher altitude available. So, by contradiction, A is minimum altitude and minimum spamnning tree as well. □

**Problem 5.** Textbook [Kleinberg & Tardos] Chapter 4, page 190, problem #25.

*Proof.* ...
Input: distance metric, d
Output: to design a hierarchical matrix, t(pi, pj) that is consistent with d.
Idea: Using Kruskal's algorithm.
Sort all the pairs (pi, pj) in increasing order of distances by connecting them. the pairs pi and pj are separately visualized as Gi and Gj where we find the maximum of heights Gi and Gj. ie d(pi, pj) = max(height(Gi), height(Gj)) and simply connect them to lower height of the root of the other component. In other case where d(pi, pj) is greater than the maximum of the heights, we create a new node representing the height of the node. Now, we make this value to root and make all compinents of Gi and Gj to be their Children. With this, we get a graph of T(Pi, Pj) to give the height of the least common ancestors. Using the contradiction theory, we can prove easily that the formalized, T(Pi, Pj) is the heirarchial component required that is maximized. □

**Problem 6.** Textbook [Kleinberg & Tardos] Chapter 4, page 190, problem #26.

*Proof.* Tried, But unable to understand the question :( □

**Problem 7.** Textbook [Kleinberg & Tardos] Chapter 4, page 190, problem #31.

*Proof.* □