CHALMERS
EXAMINATION / TENTAMEN

Course code/kurskod	Course name/kursnamn			
TIN093	Algorithms			
Anonymous code Anonym kod		Examination date Tentamensdatum	Number of pages Antal blad	Grade Betyg
TIN093-0059-SED		26-10-2022	7	5

* I confirm that I've no mobile or other similar electronic equipment available during the examination.
Jag intygar att jag inte har mobiltelefon eller annan liknande elektronisk utrustning tillgänglig under eximinationen.

Solved task Behandlade uppgifter No/nr		Points per task Poäng på uppgiften	Observe: Areas with bold contour are to completed by the teacher. Anmärkning: Rutor inom bred kontur ifylles av lärare.
1	X	10	
2	X	14	
3	X	10	
4	X	10	
5	X	12	
6			
7			
8			
9			
10			
11			

inversion if $a > b$. Once the sequence is sorted, the number of inversions must be 0. If each swap at best reduces inversions by one, it stands that at least inv swaps are needed. ✓5

1.2. Each swap in the algorithm will decrease the number of inversions by one, and will proceed until the sequence is sorted, at which point the number of inversions must be zero. As such, there must be exactly one swap in the algorithm for each inversion, and the total number of swaps is inv.

✓5

We can formulate our graph as a DAG, where the nodes are sorted by its distance from B

A ... C ... D .. B

Our algorithm can then iterate over the nodes right-to-left, in other words start with B and then pick stations further and further away from B.

For each of these nodes, we compute OPT(X) where X is the iterated node.

OPT(B) will be the desired arrival time

OPT(X) for any X except B will be computed from all edges starting in X

We filter through all edges to find those that will arrive in time, more specifically if the edge is from X to Y, we know that Y is to the right of X in the DAG ie OPT(Y) is already calculated. If the arrival time of the edge is less than OPT(Y), we keep the edge, otherwise we discard the time (because we would not arrive in time)

Of all the edges that are kept, we find the edge

edges that starts in the iterated node,
And ~~so~~ we will iterate over all nodes in polynomial time of the total number of edges.

Missing time complexity for topological sorting. No correctness. Otherwise good!

the set to one set with elements smaller than p (and thus all have rank less than r), and a set with larger elements. We then also split the set of searched ranks into those smaller than r and those larger than r. We then subtract r from all larger searched ranks and then do a recursive search on the two partitions, we continue to partition the sets until one rank is looked for in each partition. If the set is halved every time, the set of ranks to find will also halve. sets of ranks will then be at size one after around $\log_2 k$ recursions. At this point, each partition can be solved using the algorithm to find one ranked element in $O(n)$ time. Partitioning will then take $O(n \log k)$ time until we need to find one ranked element per partition, and then an additional $O(n)$

correct in polynomial time,
Uniform set packaging problems are a subset of set packaging, which means that the same polynomial algorithm can be used to verify a solution to the uniform set packaging problem. 2

4.2. We consider a reduction from set packaging to uniform set packaging.
The set packaging problem has a set of elements U, $|U| = n$, an integer k and a family of m subsets.
For each subset, we define a new element and add it to the subset until the subset has size n.
These new subsets form a new family, and we can be certain that all new subsets have the same size n.
New elements that were defined are combined with U to form U'
The new family alongside k and U' is then a uniform set packaging problem. 1

remain pairwise disjoint in the constructed problem as the reduction will only add elements that no other set has.

If there is a set of k pairwise disjoint subsets in the constructed problem, it stands that the corresponding subsets in the original problem are pairwise disjoint, since the original subsets contains no elements that are not in the constructed subsets.

The algorithm will proceed by finding any cycle accessible from s by performing a depth-first-search. If the algorithm finds a back edge (u,v), we have a path along tree edges from v to u that together with (u,v) form a directed cycle C. ok we will also have a path along tree edges from s to v which is our path P to the directed cycle. ok

If we do not come across any back edges, we can conclude that there are no directed cycles reachable from s, we could potentially still find a directed cycle elsewhere in the graph with a DFS on unvisited nodes, but these do not matter as they would be unreachable from s. ok

The time constraint of just the DFS is O(m), but that only holds if the search can utilize adjacency lists, Including the time to create these adjacency lists gives the entire algorithm a time bound of O(n+m) ok