Question 1 (Optimal Caching Exchange Argument, 25 points). In our optimal caching problem, suppose that we have a cache of size $k = 3$. Suppose that the schedule given below is used to execute the given sequences of accesses. What are the intermediate schedules produced by the exchange argument discussed in class?

**Accesses:** ABCDCBACEDABED

<table>
<thead>
<tr>
<th>Register 1</th>
<th>Register 2</th>
<th>Register 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>CD</td>
</tr>
<tr>
<td>E</td>
<td>C</td>
<td>B</td>
</tr>
</tbody>
</table>

**Solution:** Intermediate steps are as follows:

<table>
<thead>
<tr>
<th>Accesses</th>
<th>Register 1</th>
<th>Register 2</th>
<th>Register 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABCDCBACEDABED</td>
<td>A</td>
<td>B</td>
<td>CD</td>
</tr>
<tr>
<td>ABCDCBACEDABED</td>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>ABCDCBACEDABED</td>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>ABCDCBACEDABED</td>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
</tbody>
</table>

Question 2 (Rearrangement, 25 points). Given two lists $S$ and $T$ of $n$ numbers each, find a way to sort each of them $S = \{s_1, s_2, \ldots, s_n\}$ and $T = \{t_1, t_2, \ldots, t_n\}$ so that $\sum_{i=1}^{n} s_i t_i$ is as large as possible. For example, of $S = T = \{1, 2\}$, we could get either $1 \cdot 1 + 2 \cdot 2 = 5$ or $1 \cdot 2 + 2 \cdot 1 = 4$, and would prefer to find the former.

**Solution:** We can solve this problem using greedy technique.
S = MERGE-SORT(S)
T = MERGE-SORT(T)
for i = 1 to n
    sum += S[i]*T[i]
return sum

Note that this works even when the list contains negative numbers. This can be proved by arguing that any solution that does not sort the lists in the same way can be improved. In particular, if the lists are not sorted in the same order, we will have some \(i\) and \(j\) so that \(s_i > s_j\), but \(t_i < t_j\). We claim that swapping \(s_i\) and \(s_j\) will improve the total sum. To show this, we need to show that \(s_it_i + s_jt_j\) is less than \(s_jt_j + s_it_i\) (as shown below) and so the total can be improved.

Mathematically, we have to show that:

\[
s_jt_j + s_it_i > s_jt_i + s_it_j
\]

\[
= s_j(t_j - t_i) - s_i(t_j - t_i)
\]

\[
= (s_j - s_i)(t_j - t_i)
\]

> 0 (since we assumed that \(s_j > s_i\) and \(t_j > t_i\).

**Time complexity:** Sorting of each list takes \(O(n \log n)\). The next step to find the sum takes \(O(n)\). Thus, this is a \(O(n \log n)\) algorithm.

**Question 3** (Mining for Maximum Value, 25 points). Dirk is going to make money as a miner today. He has \(N\) minutes of time that he can spend on this operation and several veins of ore that he can work on. The \(i\)th vein would take \(t_i\) minutes to mine out completely, but the ore would be worth a total value of \(v_i\). Of course if Dirk spends only a fraction \(\alpha t_i\) time mining that ore (for some \(0 \leq \alpha \leq 1\)) he will get ore worth \(\alpha v_i\). Give an efficient algorithm for finding a schedule for Dirk that gets him as much total value in ore as possible and prove that it is correct.

**Solution:**

This problem is very similar to the fractional Knapsack problem and can be solved using Greedy strategy. For every vein \(i\), we are given its value \((v_i)\) and time it takes to mine out completely \((t_i)\). If Dirk spends a fraction of time \((t_i)\) at an ore \(i\), he gets the fraction of its worth. This gives Dirk the flexibility to choose the ores according to their value per unit time i.e. \((v_i/t_i)\).

The idea is to sort the list of ores in descending order (with \(v_i/t_i\) as the key) and start from the ore with largest \(v_i/t_i\) value and keep mining the ores as many possible in \(N\) minutes. Let there be \(n\) number of ores. Following algorithm takes in the value, time of each ore and the total number of minutes \((N)\), and returns the total value that Dirk can get.

**Input:** value[], time[], \(N\), n
**Output:** totalValue

**Initialize a max priority queue (PQ)**
for i = 1 to n
    Add ore i to PQ with key = value[i]/time[i]

\(T = 0\)
while \(T < N\) and PQ is not empty:
    ore = PQ.deleteMax()
    totalValue += value[ore]
    \(T += time[ore]\)
if \(T < N\) and PQ is not empty:
    ore = PQ.deleteMax()
    totalValue += \((N-T)\)*value[ore]/time[ore]
return totalValue

In order to analyze this, we first note that the order in which Dirk works on various veins does not affect anything. Therefore, we may assume that Dirk works on the veins that he does in decreasing order of
value/time. We also for simplicity assume that the ores are sorted in decreasing order of value/time. So in particular \(v_1/t_1 \geq v_2/t_2 \geq \ldots \geq v_n/t_n\).

We will now prove the correctness of our algorithm by induction. Let the greedy solution be \(G\) and any other solution be \(S\). Let \(t_k\) be the time at which \(G\) finishes mining its \(k^{th}\) vein. We prove by induction of \(k\) that at time \(f_k\) that \(G\) has mined at least as much value as \(S\). Letting \(k\) be the total number of ores mined by \(G\) will prove our result.

**Base case:** When \(k = 0\), neither \(G\) nor \(S\) have mined anything at time \(f_0 = 0\), and so they do equally well by this point.

**Inductive Step:** Assume that by time \(f_k\) that \(G\) has done at least as well as \(S\). We note that between times \(f_k\) and \(f_{k+1}\) \(G\) mines only from vein \(k + 1\), and thus gets a total value of \((f_{k+1} - f_k)v_{k+1}/t_{k+1}\). On the other hand since \(f_k = t_1 + t_2 + \ldots + t_k\), and since \(S\) mines ores in increasing order, \(S\) must have finished all mining of ores \(t_1, \ldots, t_k\) by time \(f_k\). Therefore between times \(f_k\) and \(f_{k+1}\), \(S\) mines only ores of index at least \(k + 1\). Note that these ores have a value per unit time of at most \(v_{k+1}/t_{k+1}\), and therefore, the total amount of value that \(S\) mines is at most \((f_{k+1} - f_k)v_{k+1}/t_{k+1}\). Therefore we have that

\[
G(f_{k+1}) = G(f_k) + (f_{k+1} - f_k)v_{k+1}/t_{k+1} \geq S(f_k) + (S(f_{k+1}) - S(f_k)) = S(f_{k+1}).
\]

This computes our inductive step and proves optimality.

**Time Complexity:** Extracting the ores from the priority queue can take a maximum of \(O(n \log n)\).

**Question 4** (Greedy Algorithms for Shortest Paths in DAGs, 25 points). Let \(G\) be a DAG with vertex set \(v_1, v_2, \ldots, v_n\) with edges going from \(v_i\) to \(v_j\) only for \(j > i\). Suppose that the edges of \(G\) are weighted with weights \(\ell(v_i, v_j)\), and that we want to find the shortest path from \(v_1\) to \(v_n\). For each of the proposed greedy algorithms for this problem provide a counter-example. In particular, give an explicit graph \(G\) and edge weights so that the greedy algorithm does not produce the optimal path. Show both the path produced by the greedy algorithm and the optimal one.

(a) Starting from \(v_1\) build the path one vertex at a time. From vertex \(v_i\), take the edge to \(v_j\) with \(j\) as large as possible. [5 points]

(b) Starting from \(v_1\) build the path one vertex at a time. From vertex \(v_i\), take the edge to \(v_j\) with \(\ell(v_i, v_j)\) as small as possible. [5 points]

(c) Starting from \(v_1\) build the path one vertex at a time. From vertex \(v_i\), take the edge to \(v_j\) with \(\ell(v_i, v_j)/(j - i)\) as small as possible. [5 points]

(d) Let \(S\) be a set of vertices, initially \(\{v_1\}\) and \(T\) a set of edges initially 0. Repeatedly find the edge in \(G\) from a vertex in \(S\) to a vertex not in \(S\) that has the smallest weight among such edges. Add this edge to \(T\) and add the vertex on the other end of this edge to \(S\). Repeat until \(v_n\) has been added to \(S\) and then take the unique path from \(v_1\) to \(v_n\) using only edges of \(T\). [10 points]

**Solution:**
Consider the graph given in Figure 1. For each part, we would represent the value of \(W_{ij}\) for which the greedy algorithm won’t work.

![Figure 1: Graph to consider for Question 4](image-url)
(a) $W_{12} = 1, W_{23} = 1, W_{13} = 5$. Optimal path is $V_1, V_2, V_3$. Greedy path is $V_1, V_3$.
(b) $W_{12} = 1, W_{23} = 2, W_{13} = 2$. Optimal path is $V_1, V_3$. Greedy path is $V_1, V_2, V_3$.
(c) $W_{12} = 1, W_{23} = 4, W_{13} = 4$. Optimal path is $V_1, V_3$. Greedy path is $V_1, V_2, V_3$.
(d) $W_{12} = 2, W_{23} = 2, W_{13} = 3$. Optimal path is $V_1, V_3$. Greedy path is $V_1, V_2, V_3$.

**Question 5** (Extra credit, 1 point). *Approximately how much time did you spend working on this homework?*