Name: _____

ID number: _____

There are 5 questions for a total of 15 points.

---

1. Solve the following questions:

(a) (½ point) State true or false: $7^{\log_3 10} = 10^{\log_3 7}$.

(a) **True**

Reason/Method: (*You were not expected to write this*) Take $\log_3$ of both side and check.

(b) (½ point) Recall the Longest Increasing Subsequence problem discussed in class. For the following sequence of numbers in array $A = [14, 8, 2, 7, 4, 10, 6, 0, 1, 9, 5, 13, 3, 11, 12, 15]$ give a longest increasing subsequence.

(b) **2, 4, 6, 9, 11, 12, 15**

Reason/Method: (*You were not expected to write this*) Run the algorithm discussed in class on the example above.

(c) (½ point) Give the solution (in terms of big-O notation) of the recurrence relation

$$T(n) \leq T(n/2) + \log_2 n; \quad T(1) = 1$$

(*you may assume that n is a power of* 2)

(c) $O((\log n)^2)$

Reason/Method: (*You were not expected to write this*) This follows from the following sequence of inequalities:

$$\begin{aligned}
T(n) &\leq T(n/2) + \log_2 n \\
&\leq T(n/2^2) + \log_2 n/2 + \log_2 n \\
&\vdots \\
&\leq T(n/2^{\log_2 n}) + \log_2 (n/2^{\log n - 1}) + \log_2 (n/2^{\log n - 2}) + ... + \log_2 n \\
&= T(1) + \log_2 n \cdot \log_2 n - (1 + 2 + ... + (\log_2 n - 1)) \\
&= 1 + (\log_2 n)^2 - \log_2 n(\log_2 n - 1)/2 \\
&= 1 + \frac{(\log_2 n)^2}{2} + \frac{\log_2 n}{2}
\end{aligned}$$

So, $T(n) = O((\log_2 n)^2)$.

(d) (½ point) State true or false: The Prim's algorithm always returns a minimum spanning tree of any strongly connected, weighted, undirected graph even when the graph has negative edge weights.

(d) **True**

Reason/Method: (*You were not expected to write this*) The correctness of the Prim's algorithm depends only on the relative ordering of the weight of the edges.

(e) (½ point) State true or false: For any strongly connected, weighted, undirected graph G with distinct edge weights, the maximum weight edge of G is not present in any minimum spanning tree of G.

(e) **False**

Reason/Method: (*You were not expected to write this*) Consider the following counterexample: The graph has two vertices connected with an edge with weight 10. Since this is the only edge in the graph, it is present in all MSTs of the graph and also is the edge with maximum weight.

2. Recall the Job Scheduling problem that was discussed in class. In this problem n jobs are given each with a duration and a deadline and the goal is to minimize the maximum lateness. Consider a variant of this problem where the goal is to minimize the sum of lateness of all jobs. That is, find a schedule such that the sum of lateness of jobs as per this schedule is minimized. Answer the next two questions:

   (a) (½ point) State true or false: The greedy algorithm that we discussed in class for the Job Scheduling problem also works for above variant of the problem.

   (a) **False**

   (b) (1 point) Give reason for your answer in part (a).

   Solution: Consider the following two jobs: $(1, 1)$ and $(10, 0)$. That is, the duration of the first job is 1 with a deadline 0, and the duration of the second job is 10 with a deadline 0. As per the algorithm based on the deadlines, the second job will be scheduled first. As per this schedule, the lateness of the first job is $(11 - 1) = 10$ and the lateness of the second job is $(10 - 0) = 10$. So the total lateness is 20. However, consider the schedule where the first job is schedules before the second job. In this case, the lateness of the first job is $(1 - 1) = 0$ and the lateness of the second job is $(11 - 0) = 11$, making the sum of lateness 11.

3. Given a number x and an array $A = A[1]A[2]...A[n]$ containing n distinct integers sorted in increasing order, we want to determine if x is present in the array. Assume that n is a power of 4 and consider two different divide and conquer algorithms for this task given below.

`BinarySearch`(x, A, n)
- if $(n = 1)$
  - if $(A[1] = x)$ return "yes"
  - else return "no"
- $mid \leftarrow n/2$
- $A_L \leftarrow A[1]...A[mid]$
- $A_R \leftarrow A[mid + 1]...A[n]$
- if $(x \leq A[mid])$
  - return(`BinarySearch`$(x, A_L, n/2)$)
- else
  - return(`BinarySearch`$(x, A_R, n/2)$)

`QuadrupleSearch`(x, A, n)
- if $(n = 1)$
  - if $(A[1] = x)$ return "yes"
  - else return "no"
- $mid_1 \leftarrow n/4; mid_2 \leftarrow 2n/4; mid_3 \leftarrow 3n/4$
- $A_1 \leftarrow A[1]...A[mid_1]$
- $A_2 \leftarrow A[mid_1 + 1]...A[mid_2]$
- $A_3 \leftarrow A[mid_2 + 1]...A[mid_3]$
- $A_4 \leftarrow A[mid_3 + 1]...A[n]$
- if $(x \leq A[mid_1])$
  - return(`QuadrupleSearch`$(x, A_1, n/4)$)
- elseif $(x \leq A[mid_2])$
  - return(`QuadrupleSearch`$(x, A_2, n/4)$)
- elseif $(x \leq A[mid_3])$
  - return(`QuadrupleSearch`$(x, A_3, n/4)$)
- else return(`QuadrupleSearch`$(x, A_4, n/4)$)

We would like to compare these two algorithms with respect to the number of times the search-element x is compared with an element of the array A. Let $S(n)$ denotes the number of times in the worst case x is compared with an element of A during the executing of the algorithm `BinarySearch`. Let $T(n)$ denote the number of times in the worst case x is compared with an element of A during the executing of the algorithm `QuadrupleSearch`. The recurrence relations for $S(n)$ and $T(n)$ may be written in the following general form:

$$S(n) = a \cdot S(n/b) + c; \quad S(1) = d$$
$$T(n) = p \cdot T(n/q) + r; \quad T(1) = s$$

Answer the questions that follow:

(a) (2 points) Give the values of these constants a, b, c, d, p, q, r, and s

(a) $a = 1, b = 2, c = 1, d = 1, p = 1, q = 4, r = 3, s = 1$

(b) (1 point) Solve recurrence relation for $S(n)$ and give exact value of $S(n)$.

Solution: We solve by "unrolling the recursion":

$$\begin{aligned} S(n) &= S(n/2) + 1 \\ &= S(n/2^2) + 1 + 1 \\ &= S(n/2^3) + 1 + 1 + 1 \\ &\vdots \\ &= S(n/2^{\log_2 n}) + \underbrace{1 + 1 + ... + 1}_{\log_2 n \text{ terms}} \\ &= S(1) + \log_2 n \\ &= 1 + \log_2 n \end{aligned}$$

(c) (1 point) Solve recurrence relation for $T(n)$ and give exact value of $T(n)$.

Solution: We solve by "unrolling the recursion":

$$\begin{aligned} T(n) &= T(n/4) + 3 \\ &= T(n/4^2) + 3 + 3 \\ &= T(n/4^3) + 3 + 3 + 3 \\ &\vdots \\ &= T(n/4^{\log_4 n}) + \underbrace{3 + 3 + ... + 3}_{\log_4 n \text{ terms}} \\ &= T(1) + 3 \cdot \log_4 n \\ &= 1 + 3 \cdot \log_4 n \\ &= 1 + (3/2) \cdot \log_2 n \end{aligned}$$

(d) (½ point) State true or false: $T(n) < S(n)$ for all $n \geq 256$ where n is a power of 4.
(*In other words,* `QuadrupleSearch` *is better than* `BinarySearch` *with respect to the worst-case number of times x is compared with array elements.*)

(d) **False**

What is the significance of this question or why should we be interested in the number of times x is compared with elements of the array?

In certain scenarios, the elements in the array and the element we want to search are not simple numbers but more complicated objects (such as files, images etc.). In such scenarios, the running time of the algorithm is governed by the number of comparisons.

4. (2 points) There are n men with heights $m_1, m_2, ..., m_n$ and n women with heights $w_1, w_2, ..., w_n$. You have to match men to women for a dance such that the sum of absolute value of difference in height of each pair, is minimized.
(*For example, consider $n = 2$ and $m_1 = 5, m_2 = 7, w_1 = 6.5, w_2 = 6$. In this case, if we match first man with second woman and second man with first woman, then the sum of absolute value of difference is* $|5 - 6| + |7 - 6.5| = 1.5$)

Design an algorithm to solve this problem. Give pseudocode for your algorithm and discuss running time.
(*You may assume that the heights of men and women are given in arrays M and W and your output should consist of n pairs of the form (i, j) indicating that the i^{th} man is matched with the j^{th} woman. Furthermore, you may assume that $M[1] \leq M[2] \leq ... \leq M[n]$ and $W[1] \leq W[2] \leq ... \leq W[n]$.*)

Solution: Here is the pseudocode for a greedy algorithm that works for this problem.

```
DancePairs(M, W, n)
  - S ← {}
  - for i = 1 to n
      - S ← S ∪ (i, i)
  - return(S)
```

Running time: The running time of the algorithm is $O(n)$ since it just runs a loop with n iterations.

Sketch of proof of correctness: (*you were not expected to write this*)

We will show that the following greedy algorithm minimizes the sum of absolute difference in height:

> Let the men and women be sorted in increasing order of heights. Match the first man in the sequence to the first woman, second man in the sequence to the second woman, and so on.

We will now prove optimality. As per our assumption, $m_1 \leq m_2 \leq ... \leq m_n$ and $w_1 \leq w_2 \leq ... \leq w_n$. Consider any optimal solution σ. This solution is just a permutation of $1...n$. This means that as per the optimal solution, M_1 is matched with $W_{\sigma(1)}$, M_2 is matched with $W_{\sigma(2)}$, and so on. Let i be the smallest index such that $\sigma(i) \neq i$. Let j be the index such that $\sigma(j) = i$. Clearly, $j > i$. Consider another solution σ_1 that is the same as σ, except $\sigma'(i) = i$ and $\sigma'(j) = \sigma(i)$. We will show that the average height difference as per σ' is less than equal to the average height difference as per σ. Let the average height difference as per σ and σ' be denoted by $H(\sigma)$ and $H(\sigma')$ respectively. We have:

$$H(\sigma) - H(\sigma') = \frac{1}{n} \cdot (|m_i - w_{\sigma(i)}| + |m_j - w_i| - |m_i - w_i| - |m_j - w_{\sigma(i)}|)$$

We consider the following 6 cases:

1. $w_i \leq w_{\sigma(i)} \leq m_i \leq m_j$: In this case, $H(\sigma) - H(\sigma') = 0$.
2. $w_i \leq m_i \leq w_{\sigma(i)} \leq m_j$: In this case, $H(\sigma) - H(\sigma') = (1/n) \cdot 2 \cdot (w_{\sigma(i)} - m_i) \geq 0$.
3. $m_i \leq w_i \leq w_{\sigma(i)} \leq m_j$: In this case, $H(\sigma) - H(\sigma') = (1/n) \cdot 2 \cdot (w_{\sigma(i)} - w_i) \geq 0$.
4. $w_i \leq m_i \leq m_j \leq w_{\sigma(i)}$: In this case, $H(\sigma) - H(\sigma') = (1/n) \cdot 2 \cdot (m_j - m_i) \geq 0$.
5. $m_i \leq w_i \leq m_j \leq w_{\sigma(i)}$: In this case, $H(\sigma) - H(\sigma') = (1/n) \cdot 2 \cdot (m_j - w_i) \geq 0$.

6. $m_i \leq m_j \leq w_i \leq w_{\sigma(i)}$: In this case, $H(\sigma) - H(\sigma') = 0$.

So, in all the cases $H(\sigma') \leq H(\sigma)$. We continue with this exchange operation and at the end we obtain our greedy solution.

5. Consider the following variant of the Interval Scheduling Problem. You are given n intervals. For every interval, you are given the start time, end time, and *profit* from the interval. More specifically, the i^{th} interval is specified as a triple $(s(i), f(i), p(i))$, where $s(i)$ denotes the start time, $f(i)$ denotes the finish time, and $p(i)$ denotes the profit. For any subset S of non-overlapping intervals, the total profit from this subset S is equal to the sum of profits of the intervals in S. Your goal is find the maximum profit that can be made by picking a subset of non-overlapping intervals. For simplicity, you may assume that the intervals are sorted in increasing order of their finishing time. That is, $f(1) \leq f(2) \leq ... \leq f(n)$. (*For example, consider intervals* $(1, 3, 5), (4, 6, 5), (2, 7, 11), (9, 10, 1)$. *In this case, the maximum profit that can be made is* 12 *by picking the third and fourth intervals.*)

We will solve this problem using Dynamic Programming. Let $P(i)$ denote the maximum profit that can be made by picking a subset of non-overlapping intervals from the set $\{1, ..., i\}$ of intervals.

(a) (1 point) Give a recursive formulation for $P(.)$.

Solution: For every index i, let $\pi(i)$ denote the largest index j such that (i) $j < i$ and (ii) $f(j) \leq s(i)$. If no such index exist for an index i, then $\pi(i) = 0$. We can now write the recurrence relation in terms of this mapping π.

$$\forall i > 1, P(i) = \max\{P(i-1), P(\pi(i)) + p(i)\}; \quad \text{and} \quad P(0) = 0$$

(b) (2 points) Use the above recursive formulation to design an algorithm that outputs the maximum profit that can be made. Give pseudocode.

Solution: Here is the pseudocode for the algorithm.

```
ProfitIntervalSchedule(s, f, p, n)
  - π ← ComputePi(s, f, n)
  - P[0] ← 0
  - for i = 1 to n
      - P[i] ← max{P[i − 1], P[π[i]] + p[i]}
  - return(P[n])
```

In the above pseudocode, `ComputePi` denotes the function that computes the value of $\pi(i)$ for all i where π is as per defined in part (a). Here is a pseudocode for `ComputePi`:

```
ComputePi(s, f, n)
  - for i = 1 to n
      - j ← i − 1
      - while(f[j] > s[i] and j > 0) j ← j − 1
      - if (j = 0) π[i] ← 0
      - else π[i] ← j
  - return(π)
```

(c) (½ point) Discuss running time of your algorithm.

Solution: Running time: The function `ComputePi` has two nested loops with the outer loop having n iterations and the inner loop corresponding to i has $O(i)$ iterations. So the running time of `ComputePi` is $O(n^2)$. The remaining part of the algorithm is a for loop that has n iterations costing constant amount of time per iteration, making the running time of the for loop $O(n)$. So, the overall running time of the algorithm is $O(n^2)$.

An algorithm with better running time: Note that the running time of $O(n^2)$ is governed by the `ComputePi` algorithm. We can get an algorithm for computing π with running time $O(n \log n)$, if for every index i, instead of doing a linear search for finding the largest index $j < i$ such that $f(j) \leq s(i)$, we do a binary search (we can do this since the intervals are sorted in increasing order of finish times.)

(d) (1 point) Consider the following intervals $(0, 2, 2), (1, 4, 5), (3, 5, 2), (7, 8, 2), (9, 10, 2), (6, 12, 5), (11, 13, 6)$. Fill the P table using the recursive formulation given in part (a).

	1	2	3	4	5	6	7
P	2	5	5	7	9	10	15