Practice Quiz 1

Autumn 2025

Name Answer Key

Net ID ______________ (@uw.edu)

Academic Integrity: You may not use any resources on this quiz except for your one-page (front and back) reference sheet, writing instruments, your own brain, and the exam packet itself. This quiz is otherwise closed notes, closed neighbor, closed electronic devices, etc.. The last two pages of this exam serve as a reference sheet as well as scratch work (respectively). Please detach those last two pages from the exam packet. No markings on these last two pages will be graded. Your answer for each question must fit in the answer box provided.

Instructions: Before you begin, **Put your name and UW Net ID at the top of this page.** Make sure that your name and ID are LEGIBLE. Please ensure that all of your answers appear within the boxed area provided.

(1 ESNU)**Question 1: Valley Finder - Divide and Conquer**

For this problem, you will write an algorithm that takes as input an array of doubles that's length n where $n > 4$, where this array has the following properties:

– index 0 of the array contains the value 0

– index 1 of the array contains the value -1

– index $n-1$ of the array contains the value 0

– index $n-2$ of the array contains the value -1

– no two consecutive indices of the array contain the same value.

Your goal is to write an algorithm FindValley that outputs an index i such that i is a local minimum (meaning that the value at index $i-1$ is larger than the value at index i, which is smaller than the value at index $i+1$).

Note that the properties of the input array guarantee that there is a local minimum at some index.

a) Design an algorithm for FindValley that performs only $O(\log n)$ comparison operations (i.e. a $>, <, \geq, \leq$ operation on elements in the array).

Explanation (not required for a solution)
It will be helpful to first understand why the properties of the array guarantee that a local minimum exists. The array is decreasing at the front, and increasing at the end. This means that at some point in the middle (and possibly at multiple points) it must transition from a negative slope to a positive slope. The index where that transition occurs will be a local minimum.
This can now help us to write our algorithm by checking values in the array to either find a local minimum, or else find a smaller array that is decreasing in the front and increasing at the end (again ensuring a local minimum is somewhere in between).

Sample Solution:
Input: an array of doubles that we call arr, an index i (initially 0), and an index j (initially $n-1$) where $arr[i] > arr[i+1]$ and $arr[j-1] < arr[j]$.
Goal: find a local minimum between index i and index j.
Base case: if the range from i to j has 5 or fewer elements, then one of those 5 must be a local minimum. For each element, check if it is less than both of its neighbors. Return the first element seen with this property.
Divide: Query the middle index of the range (call this $mid = \frac{i+j}{2}$), and look up $arr[mid-1], arr[mid], arr[mid+1]$. If $arr[mid-1] > arr[mid] < arr[mid+1]$ then $arr[mid]$ is a local minimum, so return mid. Otherwise we will be in one of three cases:
If $arr[mid-1] > arr[mid] > arr[mid+1]$ (i.e. the middle three indices are decreasing), then there is guaranteed to be a local minimum between mid and j.
If $arr[mid-1] < arr[mid] < arr[mid+1]$ (i.e. the middle three indices are increasing), then there is guaranteed to be a local minimum between i and mid.
If $arr[mid-1] < arr[mid] > arr[mid+1]$ (i.e. the middle the middle index is a local maximum), then there is guaranteed to be a local minimum between both i and mid as well as mid and j, so we can just pick the subproblem from i to mid.
Conquer: Recursively solve the problem on the input identified above
Combine: return the result of the recursive call.

b) Write out the recurrence that describes its number of comparisons and indicate how that yields the required bound on the number of tests.

Explanation (not required for a solution)
The divide step will do a constant number of comparisons (since it's essentially sorting 3 elements). After dividing, we will have a new region that is roughly half the size since mid was chosen to be half way between i and j. We then conquer on just one subproblem. Finally, combining requires 0 comparisons.

Sample Solution:
$T(n) = T(\frac{n}{2}) + O(1)$
we can apply the master theorem with $a = 1, b = 2, k = 0$. This means $log_b(a) = 0 = k$ giving us a running time $n^k \log(n) = n^0 \log(n) = \log(n)$.

(1 ESNU)**Question 2: Asymptotic Analysis**

a) Give a value for n_0 and c that shows $3n^3 - 2n$ belongs to $O(n^3)$. Justify why it works. (Hint: see the definition of O in the reference sheet at the end of the quiz.)

Sample Solution:
We need $3n^3 - 2n \leq cn^3$

$$\begin{aligned} 3n^3 - 2n &\leq cn^3 \\ 3n^2 - 2 &\leq cn^2 \\ 3n^3 &\leq cn^3 + 2 \end{aligned}$$

so let $c = 3$ and $n_0 = 1$

b) What is the running time of a recursive algorithm for a problem that reduces the problem on inputs of size n to 8 subproblems of the same type of size $n/2$ plus $\Theta(n^3)$ nonrecursive work? First express the running time as a recurrence relation, then give an asymptotic bound using the master theorem.

Sample Solution:
$T(n) = 8T(\frac{n}{2}) + n^3$
After applying the master theorem with $a = 8, b = 2, k = 3$ we get $n^3 \log n$

(1 ESNU)**Question 3: Algorithm Correctness**
In this problem you will use loop invariants to show that that the following algorithm correctly finds the sum of the largest two elements in a list.

```
1   sumLargestTwo(list):
2       if(list.length == 0):
3           return 0
4       if(list.length == 1):
5           return list[0]
6       temp0 = list[0]
7       temp1 = list[1]
8       first = max(temp0, temp1)
9       second = min(temp0, temp1)
10      for(i = 0; i < list.length; i++):
11          if(list[i]>first):
12              second = first
13              first = list[i]
14          else if(list[i]>second):
15              second = list[i]
16      return first + second
```

a) Provide the loop invariant that you will use to demonstrate correctness

Sample Solution:
after iteration i of the loop, the variable first contains the maximum from the first i elements and the variable second contains the next maximum from the first i elements.

b) Show that your loop invariant holds

Sample Solution:
Case 1: list[i] is neither the max nor second largest of the first i elements of list. In this case, the max from the first i elements and the second largest remain unchanged compared to that from the first i-1 elements. By assumption our claim held for iteration i-1, so because the values of first and second did not change in iteration i of the code, they are correct for iteration i as well.
case 2: list[i] is the second largest. In this case the max element from the first i indices of the list remains unchanged. By assumption our claim held for iteration i-1, so because the value of first did not change in iteration i, it still has the correct max. The second largest should now become list[i], which is done in the code on line 15.
case 3: list[i] is the max

c) Show that your loop invariant ensures that the algorithm returns the correct value.

Sample Solution:
After all i iterations, first is the overall max element and second is the overall second largest element. By returning the sum of them, the algorithm returns the sum of the largest two elements.

d) What other two things need to be shown in order to guarantee algorithm correctness (you do not need to prove these, just name or describe them)?

Sample Solution:
soundness, termination

(1 ESNU)**Question 4: Stable Matchings**

Consider the preference lists below for groups A and B.

Group A	Group B
a_1 : b_1, b_2, b_3, b_4	b_1 : a_2, a_3, a_1, a_4
a_2 : b_2, b_3, b_1, b_4	b_2 : a_3, a_1, a_2, a_4
a_3 : b_3, b_1, b_2, b_4	b_3 : a_1, a_3, a_2, a_4
a_4 : b_1, b_2, b_4, b_3	b_4 : a_1, a_2, a_3, a_4

1. Give the stable matching produced by the Gale-Shapley algorithm when group A are the proposers.

   Sample Solution:
   $(a_1, b_1)(a_2, b_2)(a_3, b_3)(a_4, b_4)$

2. Give the stable matching produced by the Gale-Shapley algorithm when group B are the proposers.

   Sample Solution:
   $(a_1, b_3)(a_2, b_1)(a_3, b_2)(a_4, b_4)$

3. Justify that EVERY stable matching matches a_4 with b_4. (Hint: consider the proposer optimality and receiver pessimality properties of Gale-Shapley.)

   Sample Solution:
   When group A are the proposers, Gale-Shapley produces a proposer-optimal matching, and so every member of group A is paired with their most prefered valid match from group B. When group B are the proposers, Gale-Shapley produces a receiver-pessimal matching, and so every member of group A is paired with their least preferred valid match from group B. Since a_4 is paired with b_4 in both, that must mean that a_4's only valid pair is with b_4, so they must be paired in every stable matching

Reference

Nothing written on this page will be graded.

Logs

$$x^{\log_x(n)} = n$$
$$\log_a(b^c) = c\log_a(b)$$
$$a^{\log_b(c)} = c^{log_b(a)}$$
$$\log_b(a) = \frac{\log_d(a)}{\log_d(b)}$$

Asymptotic Notation

$f(n)$ is $O(g(n))$ provided that after some input size n_0, $f(n) \leq c \cdot g(n)$ for some constant c.

$f(n)$ is $\Omega(g(n))$ provided that after some input size n_0, $f(n) \geq c \cdot g(n)$ for some constant c.

$f(n)$ is $\Theta(g(n))$ provided that $f(n)$ is $O(g(n))$ and $f(n)$ is $\Omega(g(n))$

Master Theorem

Suppose that $T(n) = aT(\frac{n}{b}) + O(n^k)$ for $n > b$. Then:

– if $a < b^k$ then $T(n)$ is $O(n^k)$

– if $a = b^k$ then $T(n)$ is $O(n^k \log n)$

– if $a > b^k$ then $T(n)$ is $O(n^{\log_b a})$

Proposer Optimality / Receiver Pessimality

A pair (p, r) is a valid pair if there is some stable matching where they are matched together

Proposer Optimality: Every proposer is matched with their most preferred valid pair.

Receiver Pessimality: Every receiver is matched with their least preferred valid pair.

Scratch Work

Nothing written on this page will be graded.