Topic 2b: Divide & Conquer and Tradeoffs, Sec 1 Group __

Group Members Present: Firstname Lastname <uhemail@hawaii.edu>

- Name
- Name
- Name
- Name

Come to agreement as a group on the responses to these questions, and insert your responses below each question. Everyone named above will get the same grade for this work. If there is a disagreement, you may note the alternative solutions in this document.

1. Correctness of Binary Search

Consider the Binary Search algorithm shown below (a "Decrease & Conquer" algorithm). The inputs are a ***sorted*** array *A* of integers. You will use a loop invariant to show that the algorithm finds *x* correctly if it is in the range of *A* to be searched.

```
Binary-Search(x, A, min, max)
1       low = min
2       high = max
3       while low <= high
4               mid =⌊(low + high) / 2⌋
5               if x < A[mid]
6                       high = mid - 1
7               else if x > A[mid]
8                       low = mid + 1
9               else return mid         // x == A[mid] by elimination of alternatives
10      return "NOT FOUND"
```

a. State the loop invariant for the while loop. *Hint: Do <u>not</u> summarize what the algorithm does; state a specific condition that remains true at every iteration.*

b. (**Initialization**) Referencing the code, show the invariant is true at loop initialization:

c. (Maintenance) Referencing the code, show that if the invariant is true at the beginning of a loop pass it is true at the end of the loop pass (there will be three cases):

d. (Termination) Use the truth of the invariant at exit to show that the algorithm is correct.

e. Which part of your proof fails if the array is not sorted: the Initialization, Maintenance or Termination, and why?

2. Tradeoffs in Search and Sorting

We know from the previous class work that Linear Search is $\Theta(n)$, so one might think it is always better to use the $\Theta(\lg n)$ BinarySearch. However, in order to apply Binary Search we have to sort the data, which (we will soon learn) requires $\Theta(n \lg n)$ time for the best known algorithms (e.g., MergeSort) in the general case. This question explores when it is worth paying the extra cost of sorting the data in order to apply fast Binary Search.

Suppose you will be **searching a list of *n* items *m* times**. When is *m* big enough relative to *n* to make it worth sorting and using binary search rather than just using linear search? You will answer this question below.

We have two alternatives. Simply using Linear Search *m* times on *n* items has an expected (average) cost of $\Theta(mn)$. The second alternative is (a) below, and (b)-(d) explore the tradeoffs.

a. What is the expected cost to apply Merge Sort once to sort *n* items, and then apply Binary Search *m* times to *n* items?

b. Suppose $m = 1$: which strategy is better, and why? Use the above expressions to justify your answer mathematically.

Linear Search time:
Hybrid Search Time:
The Winner:

c. Suppose $m = n$: which strategy is better, and why? Use the above expressions to justify your answer mathematically.

Linear Search time:
Hybrid Search Time:
The Winner:

d. What is the cutoff point in terms of *m* expressed as a function of *n* between when it is

faster to just apply the linear search and when it is faster to apply Merge Sort and Binary Search? Set up an equation and solve for m. (For parts d and e we pretend that the formulas are exact, and leave off Θ.)

e. Using your formula, and assuming a data size of $n=2^{10}=1024$ items, for what value of m does it become worth sorting the data? What about $n=2^{20}=1048576$ items?

Challenge Problems:

1) Use a loop invariant to prove that your Linear Search is correct. (This will be on the homework.)
2) How would you use recursion trees to analyze the time complexity of Binary Sort?