CECS 328 Quiz 3

Darin Goldstein

The Mergesort algorithm works as follows.
Let arr be an array of integers.

```
MERGESORT($arr$):
  if $|arr| = 1$ then
    return arr
  end if
  Split $arr$ into two (roughly) equal-length arrays: $left$ and $right$
  $leftSorted \leftarrow MERGESORT(left)$
  $rightSorted \leftarrow MERGESORT(right)$
  $sorted = MERGE(leftSorted, rightSorted)$
  return $sorted$
```

Where MERGE($left, right$) merges the two arrays into one sorted array, in linear time.

1

Let $n = |arr|$. Which of the following represents the running time for Mergesort?

(a) $T(n) = 3T(\frac{n}{2}) + O(n)$

(b) $T(n) = 2T(\frac{n}{3}) + O(n)$

(c) $T(n) = 2T(\frac{n}{2}) + O(n)$

(d) $T(n) = 4T(\frac{n}{2}) + O(n^2)$

(e) None of the above

2

Use the Master Theorem on your recurrence relationship from Question 1. How fast does $T(n)$ grow for the Mergesort algorithm?

(a) $\Theta(n)$

(b) $\Theta(n \log(n))$

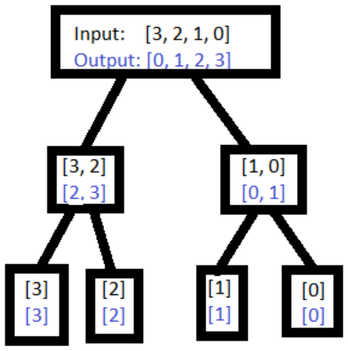
(c) $\Theta(n^{\log_2(3)})$

(d) The Master Theorem cannot be used on this recurrence.

(e) None of the above

3

Model the outputs of an instance of the Mergesort algorithm as a binary tree, where each node is the sorted output for that recursion level.
For example, if sorting arr = [3, 2, 1, 0], then the 0th level (root) output of the tree is [0, 1, 2, 3], the 1st level outputs left node [2, 3] and right node [0, 1], etc.



For the array $[7, -1, -1, -10, 3, -4, -7, 4]$, what output arrays will you find on the 2nd level of this tree?

(a) $[7, -1, -1, -10], [3, -4, -7, 4]$

(b) $[7, -1], [-1, -10], [3, -4], [-7, 4]$

(c) $[7], [-1], [-1], [-10], [3], [-4], [-7], [4]$

(d) $[-1, 7], [-10, -1], [-4, 3], [-7, 4]$

(e) None of the above

4

The Binary Search algorithm works as follows.
Let *arr* be a sorted array of integers, and say we are searching for the index of

a value toFind.
(Notation: Let arr[i:j] represent the sub-array of *arr* from indices i to j)

```
BIN_SEARCH($arr$, toFind):
  **if** $|arr| = 0$ **then**
    **return** -1 (Failed Search)
  **end if**
  $n \leftarrow length(arr)$
  $mid \leftarrow \lfloor n/2 \rfloor$
  **if** $toFind < arr[mid]$ **then**
    BIN_SEARCH($arr[0 : mid - 1]$)
  **else if** $toFind < arr[mid]$ **then**
    BIN_SEARCH($arr[mid + 1 : n - 1]$)
  **else**
    **return** mid
  **end if**
```

Which of the following represents the asymptotic running time for binary search?

(a) $T(n) = 2T(\frac{n}{2}) + O(1)$

(b) $T(n) = 2T(\frac{n}{2}) + O(n)$

(c) $T(n) = T(\frac{n}{2}) + O(n)$

(d) $T(n) = T(\frac{n}{2}) + O(1)$

(e) None of the above

5

Use the Master Theorem on your recurrence relationship from the previous question. How fast does $T(n)$ grow for the binary search algorithm?

(a) $\Theta(1)$

(b) $\Theta(n)$

(c) $\Theta(n \log(n))$

(d) The Master Theorem cannot be used on this recurrence.

(e) None of the above

6

If given the sorted array [1, 2, ... , 8] and searching for 7, how many total calls to BIN_SEARCH will be made?

(a) 1 call

(b) 2 calls

(c) 3 calls

(d) 4 calls

(e) None of the above

7

If given the sorted array [1, 2, ... , 2048] and searching for 2048, about how many total calls to BIN_SEARCH will be made?

(a) 8 calls

(b) 11 calls

(c) 14 calls

(d) 17 calls

(e) None of the above

8

Radix sort works as follows.
Given an array of n elements, with the largest element being d digits long:
For every digit, starting from the lowest digit on up, sort the numbers in-place.

Example: **[1, 14, 50, 75, 100, 2431]** (So n = 6, d = 4)
We'll underline the digit that was just sorted on.
STEP 1: [5$\underline{0}$, 10$\underline{0}$, $\underline{1}$, 243$\underline{1}$, 1$\underline{4}$, 7$\underline{5}$]
STEP 2: [1$\underline{0}$0, $\underline{0}$1, $\underline{1}$4, 24$\underline{3}$1, $\underline{5}$0, $\underline{7}$5]
STEP 3: [$\underline{0}$01, $\underline{0}$14, $\underline{0}$50, $\underline{0}$75, $\underline{1}$00, 2$\underline{4}$31]
STEP 4: [$\underline{0}$001, $\underline{0}$014, $\underline{0}$050, $\underline{0}$075, $\underline{0}$100, $\underline{2}$431]

Note that the digit-by-digit sort subroutine is typically implemented with an O(n) algorithm called Counting Sort which, for simplicity's sake, we won't cover here. For the purposes of this quiz, it is enough to know its asymptotic running time.

What is the running time of Radix Sort? Is this better than other sorting algorithms you've seen?

(a) $O(n)$

(b) $O(d)$

(c) $O(nd)$

(d) $O(n+d)$

(e) None of the above

9

Say there are n non-negative integers in an array, and they are all unique. We want to sort them via Radix Sort. At least how many bits are needed to represent every number (stated in the language of the problem, what is a lower bound on d)?

(a) $\Omega(\log(n))$

(b) $\Omega(n)$

(c) $\Omega(n\log(n))$

(d) $\Omega(n^2)$

(e) None of the above

10

Using the information you've uncovered with the last two problems, what is the running time of radix sort on an array of arbitrary nonnegative integers? How does this compare to other sorting algorithms you've seen?

(a) $\Omega(\log(n))$

(b) $\Omega(n)$

(c) $\Omega(n\log n)$

(d) $\Omega(n^2)$

(e) None of the above