PROBLEM SOLVING:

Sorting and the growth of functions

CS340

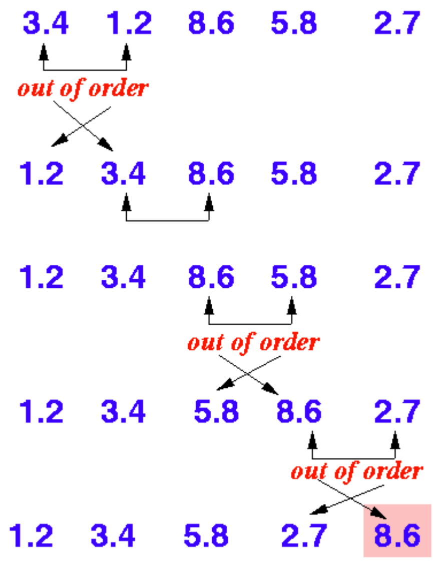
Interview Questions

```
LINEAR-SEARCH(A, v)
for i = 1 to A.length
    if A[i] == v
        return i
return NIL
```

- Consider **linear search**.
  - **How many elements** of the input sequence **need to be checked** on the **average**, assuming that the element being searched for is equally likely to be any element in the array?
  - How about in the **worst** case?
  - How about in the **best** case?
  - What are the **average-case** and **worst-case** running times of linear search in Θ-notation?
  - What if we know something about the data: eg, it's sorted?
- How can we modify **almost any algorithm** to have a good best-case running time?

Bubble Sort

```
void bubbleSort(int arr[], int n)
{
  int i, j;
  for (i = 0; i < n; i++)

     // Last i elements are already in place
     for (j = 0; j < n-i-1; j++)
        if (arr[j] > arr[j+1])
            swap(&arr[j], &arr[j+1]);
}
```



Next slide: time complexity!

What is the time complexity of bubble sort?

1. $\Theta(n)$
2. $\Theta(n \lg n)$
3. $\Theta(n^2)$
4. $\Theta(n_3)$
5. None of the above

Bubble Sort

Questions: Remember Insertion Sort had a best and worst case.

1. What is the worst case for Bubblesort?

2. How about the best case?

3. Can we change Bubblesort so that it has a better best case?

```
void bubbleSort(int arr[], int n)
{
  int i, j;
  for (i = 0; i < n; i++)

     // Last i elements are already in place
     for (j = 0; j < n-i-1; j++)
        if (arr[j] > arr[j+1])
           swap(&arr[j], &arr[j+1]);
}
```

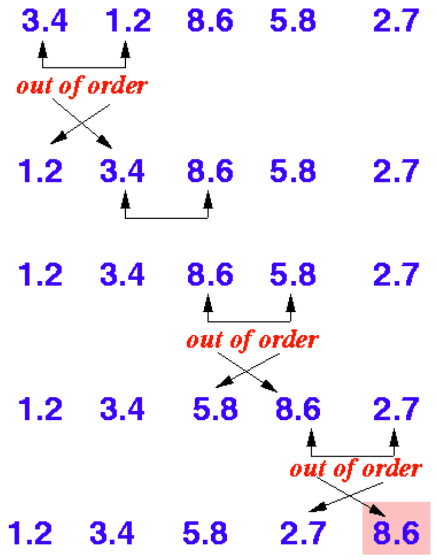
Bubble Sort

Questions:

1. What is the loop invariant for Bubblesort?

2. What are proofs for initialization, maintenance and termination?

```
void bubbleSort(int arr[], int n)
{
  int i, j;
  for (i = 0; i < n; i++)

     // Last i elements
     //are already in place
     for (j = 0; j < n-i-1; j++)
        if (arr[j] > arr[j+1])
            swap(&arr[j], &arr[j+1]);
}
```



Find the Largest Element in an Array

1. What is the time complexity?
2. What is best case?
3. What is worst case?
4. What is the loop invariant for Largest?
5. What are proofs for initialization, maintenance and termination?
6. What if we know the data are sorted?
7. What if we want to find the 2 largest elements?
8. What if we want to find the largest k elements? What's another possible algorithm?

```
int largest(int arr[], int n)
{
    int i;
    // Initialize maximum element
    int max1 = arr[0];
    // Traverse array elements
    // from second and compare
    // every element with current max
    for (i = 1; i < n; i++)
        if (arr[i] > max1)
            max1 = arr[i];
    return max1;
}
```

Finding the Median

1. What is the median?
2. What is an algorithm for finding the median?
3. What is the time complexity of the proposed algorithm?

Interview Questions

- Consider sorting n numbers stored in array A by first finding the largest element of A and exchanging it with the element in A[n]. Then find the second largest element of A and exchange it with A[n-1]. Continue in this manner for n iterations.

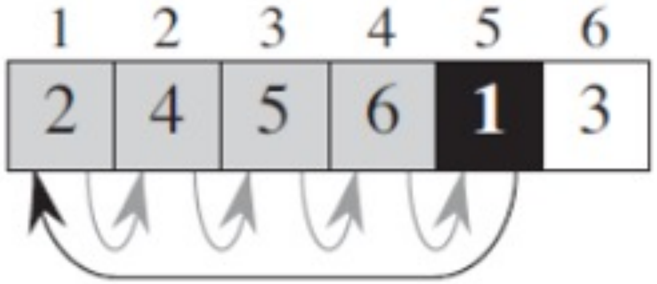
- What loop invariant does this algorithm maintain?
- How could we use this sort to compute a median?
  - Give answer on next slide!

Is the time complexity of using this sort improved as opposed to using insertion sort?

1. Yes, this sort takes half the time because we stop at n/2 swaps
2. Yes, this sort is only O(n lg n) time complexity
3. No, both approaches have O(n^2) time complexity
4. No, this algorithm cannot find the median
5. None of the above

Binary-Search enhanced Insertion Sort

- Insertion Sort spends a lot of time looking backwards to find the insertion point of a number.
- How about using binary search to find the insertion point of the key instead?
- (Question on next slide)



Does using binary search to find the insertion point improve the theta of insertion sort?

1. Yes, all insertions are faster because the insertion point is known
2. Yes, in the worst case fewer numbers need to be swapped
3. Yes, in the best case fewer numbers need to be swapped
4. No, in the worst case more numbers need to be swapped
5. No, it doesn’t change the numbers that need to be swapped in any case.

Interview Questions

- Describe a Θ(n lg n)-time algorithm that, given a set S of n integers and another integer x, determines whether or not there exist two elements in S whose sum is exactly x.
  - What is the "naïve" (might be called “brute force”) way of doing this? (and its theta?)
  - What do you think of when you hear Θ(n lg n)?

Two numbers equal X

-
  ```
  public static boolean twoNumbersEqualX(int[] A, int X) {
          A.sort();
          int start = 0;
          int end = A.length - 1;
          while (start < end) {
              int result = A[start] + A[end];
              if (result == X) return true;
              else if (result > X) end--;
              else start++;
          }
          return false;
  }
  ```
-

What technique does mergesort use to implement sorting?

1. Backtracking
2. Greedy Algorithm
3. Dynamic programming
4. Divide and conquer
5. None of the above

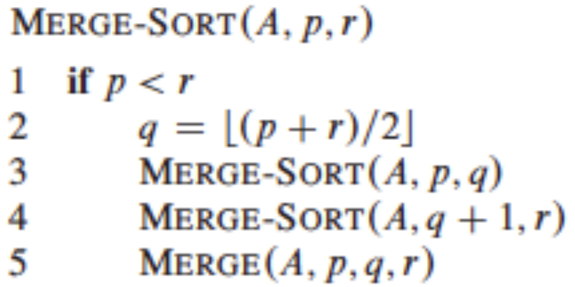
Does use of mergesort improve the theta of finding the median compared to an algorithm that uses insertion sort?

1. Yes, time complexity is improved from n^2 to n lg n
2. Yes, time complexity can be reduced to O(n) because the n/2 item merged in the first merge operation is the median
3. No, in both cases it has time complexity $O(n^2)$
4. No, because sorting the data is not a valid way to find the median
5. None of the above

Mergesort to Largest??

1. Can we use an algorithm like Mergesort to find the largest element in an array?
2. What is the code?

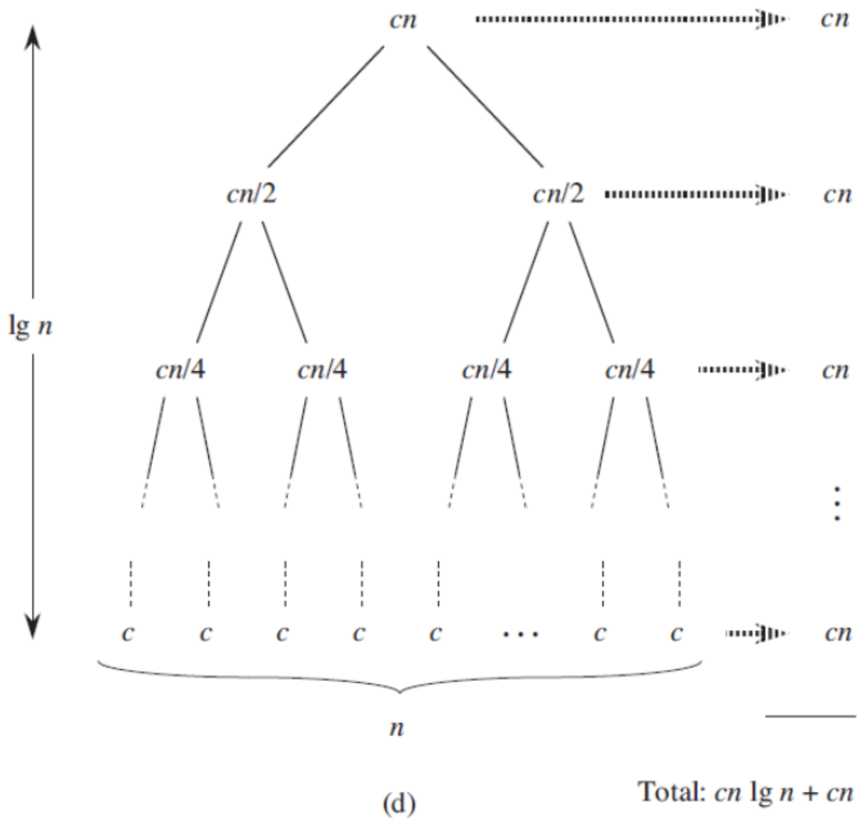
```
Int largest (A, p, r) {

}
```



```
MERGE-SORT(A, p, r)
if $p < r$
    $q = \lfloor (p + r)/2 \rfloor$
    MERGE-SORT(A, p, q)
    MERGE-SORT(A, q + 1, r)
    MERGE(A, p, q, r)
```

Mergesort to Largest

- Draw a recursion tree of the divide-and-conquer algorithm to find the largest element in an array.
- What is its time complexity?
- What is the secret to a low time complexity?
- Does this work if we are trying to find the second largest element?



Mergesort

- How could you adjust Mergesort so that it could be used in situations where you are short of memory?

Interview Questions

Express $n^3/1000 - 100n^2 - 100n + 3$ in terms of Θ notation.

Interview Questions

Although merge sort runs in $\Theta(n \lg n)$ worst-case time and insertion sort runs in $\Theta(n^2)$ worst-case time, the constant factors in insertion sort can make it faster in practice for small problem sizes on many machines. Thus, it makes sense to coarsen the leaves of the recursion by using insertion sort within merge sort when subproblems become sufficiently small. Consider a modification to merge sort in which n/k sublists of length k are sorted using insertion sort and then merged using the standard merging mechanism, where k is a value to be determined.

- Show that insertion sort can sort the n/k sublists, each of length k, in $\Theta(nk)$ worst-case time.
- b. Show how to merge the sublists in $\Theta(n \lg n/k)$ worst-case time.
- c. Given that the modified algorithm runs in $\Theta(nk + n \lg(n/k))$ worst-case time, what is the largest value of k as a function of n for which the modified algorithm has the same running time as standard merge sort, in terms of Θ-notation?
- d. How should we choose k in practice?

Interview Questions

- Consider the problem of adding two n-bit binary integers, stored in two n-element arrays A and B. The sum of the two integers should be stored in binary form in an n + 1 - element array C. State the problem formally and write pseudocode for adding the two integers.