CSE 12 – Basic Data Structures and Object-Oriented Design
Lecture 14

Greg Miranda and Paul Cao, Winter 2021

This lecture is being recorded

Announcements

- Quiz 14 due Wednesday @ 8am
- PA5 due Wednesday @ 11:59pm
- Survey 6 due Friday @ 11:59pm

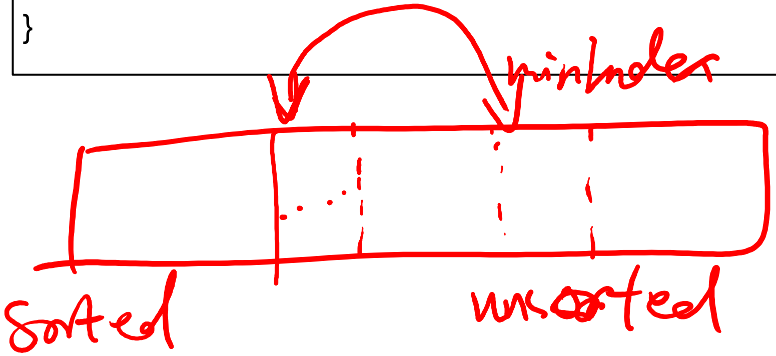
Topics

- Sorting Wrap-up
- Questions on Lecture 14?

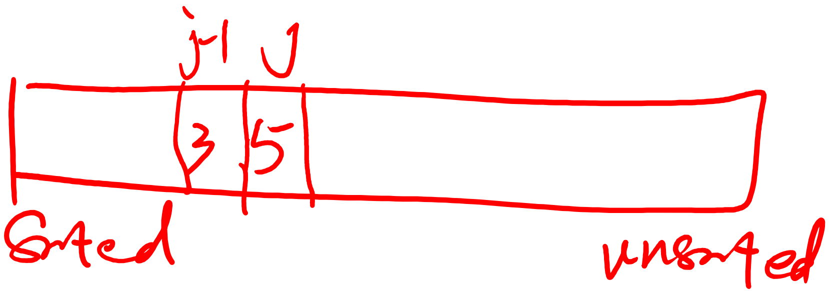
Questions on Lecture 14?

```java
import java.util.Arrays;
public class Sort {
static void selectionSort(int[] arr) {
  for(int i = 0; i < arr.length; i += 1) {
    int minIndex = i;
    for(int j = i; j < arr.length; j += 1) {
      if(arr[minIndex] > arr[j]) { minIndex = j; }
    }
    int temp = arr[i];
    arr[i] = arr[minIndex];
    arr[minIndex] = temp;
  }
}
```

loop invariant




```java
static void insertionSort(int[] arr) {
  for(int i = 0; i < arr.length; i += 1) {
    for(int j = i; j > 0; j -= 1) {
      if(arr[j] < arr[j-1]) {
        int temp = arr[j-1];
        arr[j-1] = arr[j];
        arr[j] = temp;
      }
      else { break; } // new! exit inner loop early
    }
  }
}
}
```



top - down

```java
import java.util.Arrays;
public class SortFaster {

  static int[] combine(int[] p1, int[] p2) {...}

  static int[] mergeSort(int[] arr) {
    int len = arr.length
    if(len <= 1) { return arr; }
    else {
      int[] p1 = Arrays.copyOfRange(arr, 0, len / 2);
      int[] p2= Arrays.copyOfRange(arr, len / 2, len);
      int[] sortedPart1 = mergeSort(p1);
      int[] sortedPart2 = mergeSort(p2);
      int[] sorted = combine(sortedPart1, sortedPart2);
      return sorted;
    }
  }
```

divide lgn layers

n

merge lgn layers

n

2nlgn

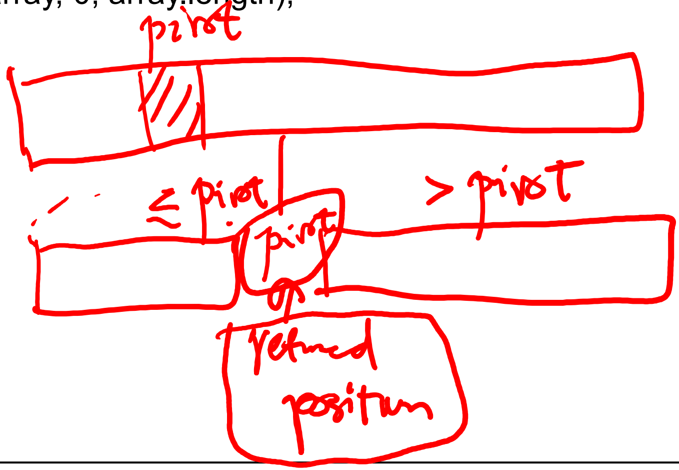

```java
static int partition(String[] array, int l, int h) {...}

static void qsort(String[] array, int low, int high) {
  if(high - low <= 1) { return; }
  int splitAt = partition(array, low, high);
  qsort(array, low, splitAt);
  qsort(array, splitAt + 1, high);
}

public static void sort(String[] array) {
  qsort(array, 0, array.length);
}

}
```

location



Quick sort

low

high

sort {12, 4, 9, 3, 15, 8, 19, 2}

i

j

pivot: 2

```
Quicksort(Arr, low, high)
  if (low < high)
    pivotPos = partition(Arr, low, high)
    Quicksort(Arr, low, pivotPos - 1)
    Quicksort(Arr, pivotPos + 1, high)
```

0

$\boxed{2}$ 4 9 3 15 8 19 12

i

return 0 (location of 2)

```
partition(Arr, low, high)
  pivot = Arr[high]
  i = low - 1
  for (j = low; j < high, j++){
    if (Arr[j] <  pivot)
      i++
      swap(A[i], A[j])
  swap(A[i+1], A[high])
  return i + 1
```

pivot Pos is always the median

$$f(x) = x + \frac{1}{x}$$

$$T(n) = T(1^{st}) + T(2^{nd}) + T(\text{partition})$$

	Insertion	Selection	Merge ave $O(n \lg n)$	Quick ave $O(n \lg n)$
Best case time	$O(n)$	$O(n^2)$	$O(n \lg n)$	$O(n \lg n)$
Worst case time	$O(n^2)$	$O(n^2)$	$O(n \lg n)$	1st 2nd $O(n^2)$
Key operations	swap(a, j, j-1) (until in the right place)	swap(a, i, indexOfMin) (after finding minimum value)	l = copy(a, 0, len/2) r = copy(a, len/2, len) ls = sort(l) rs = sort(r) merge(ls, rs)	p = partition(a, l, h) sort(a, l, p) sort(a, p + 1, h)

n
$n-1$
$n-2$
$\vdots$
1

$\frac{n(n+1)}{2}$

$O(n^2)$

Non-comparison based sorting

- Normally it is for integer sorting
- Count sort
  - Assume that we have n positive integers and we know that all of them are <=k

(know the range)

For comparison-based sorting, best you can do $O(n \lg n)$ in the worst case

$O(n+k)$

linear sorting algorithm

Count sort (A, B, k):

```
let C[0,...,k] be a new array and initialized to be 0
for j = 1 to A.length
  C[A[j]]++
for i = 1 to k
  C[i] = C[i] + C[i-1]
for j = A.length down to 1
  B[C[A[j]]] = A[j]
  C[A[j]] = C[A[j]] - 1
```

auxiliary array (can be big)

summary freq

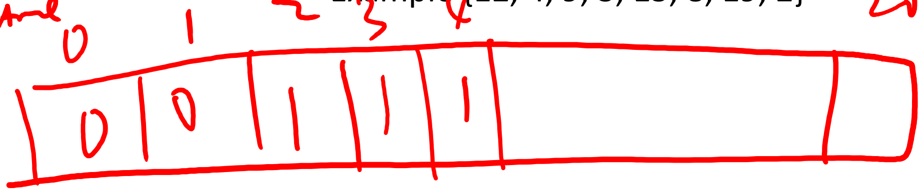
culmulative freq

fill in the sorted array

A

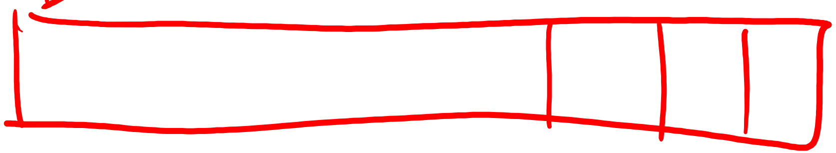
k: 20

Example {12, 4, 9, 3, 15, 8, 19, 2}



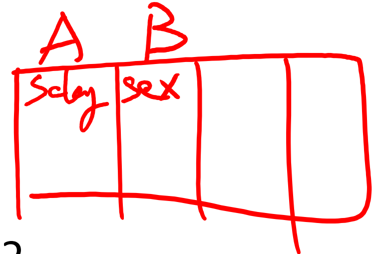
$O(k+n)$

B



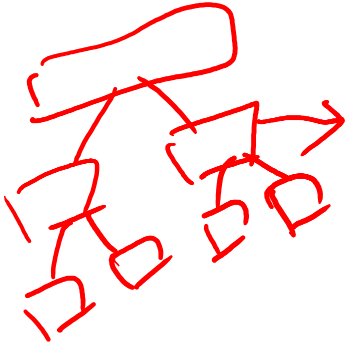
Last note about sorting

space - time - tradeoff



- Not only do we care about runtime, we also care about
  - Space: do we need extra storage?
  - stable: if we have duplicates, do we maintain the same ordering?

Algorithm	Space	Stable
Bubble sort	O(1)	Yes
Selection sort	O(1)	No
Insertion sort	O(1)	Yes
Heap sort	O(1)	No
Merge sort	O(n)	Yes
Quick sort	O(logn)	No
Count sort	O(n+k)	Yes



2_1 3 2_2 7 5

$\Downarrow$

2_1 2_2 3 5 7

Array Sorting Algorithms (wiki)

Algorithm	Time Complexity		
	Best	Average	Worst
Quicksort	Ω(n log(n))	Θ(n log(n))	O(n^2)
Mergesort	Ω(n log(n))	Θ(n log(n))	O(n log(n))
Timsort	Ω(n)	Θ(n log(n))	O(n log(n))
Heapsort	Ω(n log(n))	Θ(n log(n))	O(n log(n))
Bubble Sort	Ω(n)	Θ(n^2)	O(n^2)
Insertion Sort	Ω(n)	Θ(n^2)	O(n^2)
Selection Sort	Ω(n^2)	Θ(n^2)	O(n^2)
Tree Sort	Ω(n log(n))	Θ(n log(n))	O(n^2)
Shell Sort	Ω(n log(n))	Θ(n(log(n))^2)	O(n(log(n))^2)
Bucket Sort	Ω(n+k)	Θ(n+k)	O(n^2)
Radix Sort	Ω(nk)	Θ(nk)	O(nk)
Counting Sort	Ω(n+k)	Θ(n+k)	O(n+k)
Cubesort	Ω(n)	Θ(n log(n))	O(n log(n))