INTRO TO SORTING

cs2420 | Introduction to Algorithms and Data Structures | Spring 2015

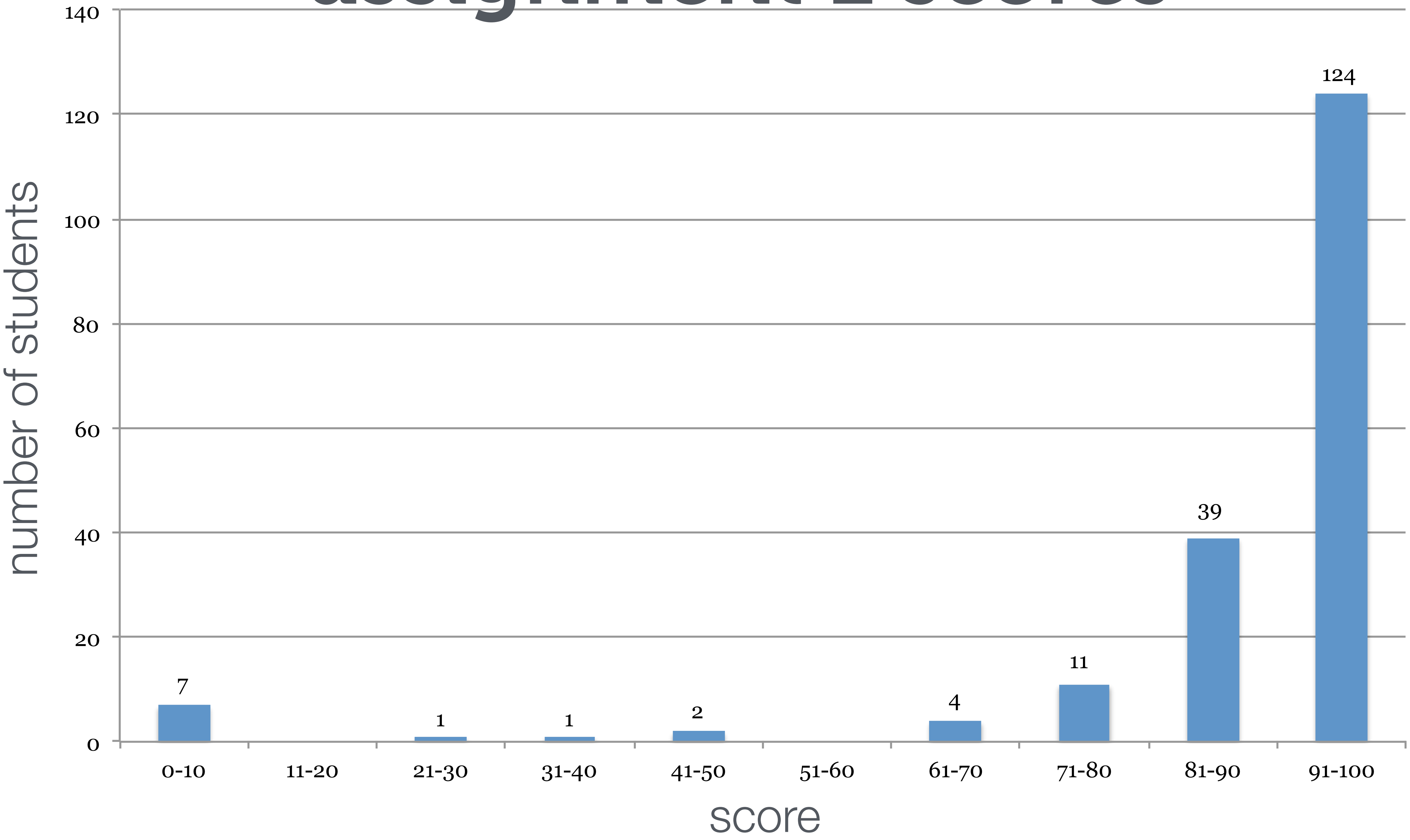
administrivia…

-assignment 3 is due Thursday at midnight (11:59pm)

-tutoring

 -Doodle polls up on the website

assignment 1 scores



CLICKERS!!!

…it’s gonna work!

Channel 41

Session ID 716193

ARE YOU HERE?

A) yes
B) no

last time…

- a `Collection` is a data structure that holds items
  - very unspecific as to how the items are held
    - *ie. the data structure*

- supports various operations:
  - `add, remove, contains`, …

- examples:
  - `ArrayList`
  - `PriorityQueue`
  - `LinkedList`
  - `TreeSet`

WHAT IF WE USE AN ARRAY UNDER THE HOOD?

add

```
int[] data = new int[6];
data.add(5);
data.add(17);
data.add(9);
data.add(12);
data.add(1);
data.add(33);
```

5	17	9	12	1	33

```
data.add(22);
```

NOW WHAT???

WHAT IS THE COMPLEXITY OF `add`?

A) **c**
B) **log N**
C) **N**
D) **N log N**
E) **N^2**
F) **N^3**

grow

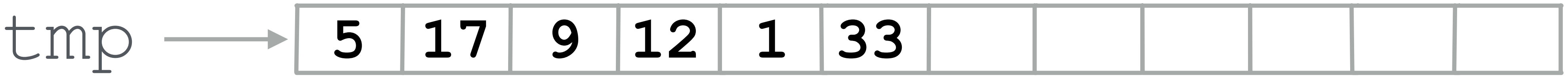
`data` →

5	17	9	12	1	33

```
tmp = new int[data.length*2];
```

`tmp` →

copy all from `data` to `tmp`



`tmp` →

5	17	9	12	1	33						

```
data = tmp;
```

`data` →

5	17	9	12	1	33						

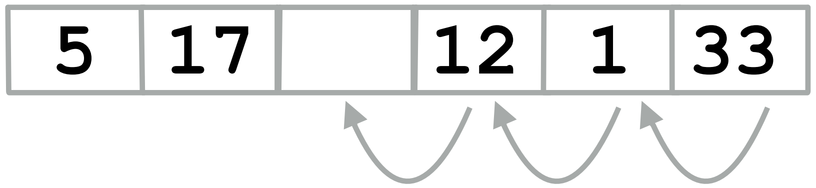
WHAT IS THE COMPLEXITY OF GROWING?

A) **c**
B) **log N**
C) **N**
D) **N log N**
E) **N^2**
F) **N^3**

remove

5	17	9	12	1	33

```
data.remove(9);
```



5	17		12	1	33

5	17	12	1	33	

WHAT IS THE COMPLEXITY OF `remove`?

A) **c**
B) **log N**
C) **N**
D) **N log N**
E) **N^2**
F) **N^3**

`Iterator`

-an `Iterator` is specific to a data structure, and knows how to traverse the structure

- -`hasNext`: determines if iteration is complete
- -`next`: gets the next item
- -`remove`: removes the las seen item

-internally, keeps track of where the next item is (as well as other state)

-actually points to ***between*** items

today…

-why sort?

-selection sort

-insertion sort

why sort?

-sorting is a fundamental application in computing

  -one of the most intensively studied and important operations

-most data is useless unless it is in some kind of order

-for any given problem, or specific goal isn’t necessarily sorting… but we often need to sort to efficiently solve problems

  -computer graphics

  -look-up tables

  -games

-sorting algorithms that are easy to understand (and implement) run in **quadratic time**

-more complicated algorithms cut it to **O(N log N)**

  -implementation details are critical to attaining this bound!

-for very specific types of data we can actually do better

  -but we won’t study these algorithms extensively

WITHOUT THINKING TOO HARD, HOW CAN WE SORT ANY ARRAY OF ITEMS?

selection sort

the simplest sorting algorithm

selection sort

1) find the minimum item in the unsorted part of the array
2) swap it with the first item in the unsorted part of the array
3) repeat steps 1 and 2 to sort the remainder of the array

WHAT DOES THIS LOOK LIKE?

```
void selectionSort(int[] arr)
{
  for(int i=0; i < arr.length-1; i++)
  {
    min = i;
    for(int j=i+1; j < arr.length; j++)
      if (arr[j] < arr[min])
        min = j;

    temp = arr[i];
    arr[i] = arr[min]
    arr[min] = temp;
  }
}
```

LAST ITEM IN SORTED PART OF ARRAY

LOOK FOR ITEM LESS THAN THOSE IN SORTED PART OF ARRAY

SWAP ITEMS

WHAT IS THE COMPLEXITY OF SELECTION SORT?

```
L1 for(int i=0; i < arr.length-1; i++)
L2   for(int j=i+1; j < arr.length; j++)
```

[redacted]

```
void selectionSort(int[] arr)
{
  for(int i=0; i < arr.length-1; i++)
  {
    min = i;
    for(int j=i+1; j < arr.length; j++)
      if (arr[j] < arr[min])
        min = j;

    temp = arr[i];
    arr[i] = arr[min]
    arr[min] = temp;
  }
}
```

- A) **c**
- B) **log N**
- C) **N**
- D) **N log N**
- E) **N^2**
- F) **N^3**

WHAT IS THE BEST-CASE COMPLEXITY OF SELECTION SORT?

insertion sort

good for small **N**

insertion sort

1) the first array item is the sorted portion of the array
2) take the second item and insert it in the sorted portion
3) repeat steps 1 and 2 to sort the remainder of the array

WHAT DOES THIS LOOK LIKE?

```
void insertionSort(int[] arr)
{
  for(int i=1; i < arr.length; i++)
  {
    index = arr[i];
    j = i;
    while(j>0 && arr[j-1]>index)
    {
      arr[j] = arr[j-1];
      j--;
    }
    arr[j] = index;
  }
}
```

ITEM TO BE INSERTED

UNTIL THE INSERTION POSITION IS FOUND, SHIFT SORTED ITEMS

INSERT ITEM

WHAT IS THE COMPLEXITY OF INSERTION SORT?

unsortedness

-requires a measure of ***unsortedness*** for array

-**inversion**: a pair of array items that are out of order

45	-3	9	76	11	-8	0

HOW MANY INVERSIONS ARE THERE?

-sorting efficiency depends on how many inversions are removed per step

insertion sort complexity

each swap to the left removes one inversion…

…we must visit each item at least once (**N**)…

…and we must undo **I** inversions

45	-3	9	76	11	-8	0

SWAP REMOVES ONE INVERSION

insertion sort is O(N+I)

HOW DO WE FIGURE OUT WHAT **I** IS?

next time…

-reading
  - -chapters 8.1 - 8.4

-homework
  - -assignment 3 due on Thursday