CMSC201
Computer Science I for Majors

Lecture 23 – Sorting

Last Class We Covered

- Searching
  - Linear search
  - Binary search
- Recursion
  - Recursion
    - Recursion

Any Questions from Last Time?

Today's Objectives

- To learn about some sorting algorithms
  - Selection Sort
  - Bubble Sort
  - Quicksort
- To start examining which of these algorithms is best for different sorting situations
  - "Run" time

“Run” Time

“Run” Time

- An algorithm’s ***run time*** is the amount of time it takes for that algorithm to run
- Run time is shown as an expression, which updates based on how large the problem is
- Run time shows how an algorithm *scales*, or changes with the size of the problem

Example: Fibonacci Recursion

- Ideally, we want an algorithm that runs in a reasonable amount of time, no matter how large the problem

- Remember the recursive Fibonacci program?
  - It runs within one second for smaller numbers
  - But the larger the number we ask for, the longer and longer it takes

Fibonacci Recursion

```
python fibEx.py (with num < 30):
   < 1 second

python fibEx.py (with num = 30):
   2 seconds

python fibEx.py (with num = 35):
   8 seconds

python fibEx.py (with num = 40):
   76 seconds
```

Fibonacci Recursion

`python fibEx.py (with num = 50):`

`Guess!`

`9,493 seconds`

`2 hours, 38 minutes, 13 seconds!!!`

Run Time for Linear Search

- Say we have a list that <u>does not</u> contain what we're looking for.
- How many things in the list does linear search have to look at for it to figure out the item's not there for a list of 8 things?
- 16 things?
- 32 things?

Run Time for Binary Search

- Say we have a list that <u>does not</u> contain what we're looking for.
- What about for binary search?
  - How many things does it have to look at to figure out the item's not there for a list of 8 things?
  - 16 things?
  - 32 things?
- Notice anything different?

Different Run Times

- These algorithms scale differently!
  - Linear search does work equal to the number of items in the list
  - Binary search does work equal to the **`log`**$_2$ of the numbers in the list!
- A **`log`**$_2$**`(x)`** is basically asking "2 to what power equals x?"
  - This is the same as saying, "how many times must we divide x in half before we hit 1?"

Sorting

Sorting Algorithms

- Sorting algorithms put the elements of a list in a specific order

- A sorted list is necessary to be able to use certain other algorithms
- Like binary search!
  - If sorted once, we can search many, many times

Sorting Algorithms

- There are many different ways to sort a list
- What method would you use?

- Now imagine you can only look at ***at most*** two elements at a time
  - What method would you use now?

- Computer science has a number of commonly used sorting algorithms

Selection Sort

Selection Sort Algorithm

- Here is a simple way of sorting a list:

1. Find the smallest number in a list
2. Move that to the end of a new list
3. Repeat until the original list is empty

Selection Sort Run Time

- What is the run time of finding the lowest number in a list?

- For a list of size **`N`**, what is the worst case number of elements you'd have to look through to find the min?
- **`N`**

Selection Sort Run Time

- For a list of size **N**, how many times would we have to find the min to sort the list?
- **N**

- What is the run time of this sorting algorithm?
- $\mathbf{N}^2$

Selection Sort Video

Video from https://www.youtube.com/watch?v=Ns4TPTC8whw

Bubble Sort

Bubble Sort Algorithm

- Let’s take a look at another sorting method!

1. We look at the first pair of items in the list, and if the first one is bigger than the second one, we swap them
2. Then we look at the second and third one and put them in order, and so on
3. Once we hit the end of the list, we start over at the beginning
4. Repeat until the list is sorted!

Bubble Sort Example

```
[ 4, 8, 1, 10, 13, 14, 6]
```

First pass:

4 and 8 are in order

8 and 1 should be swapped:

```
[ 4, 1, 8, 10, 13, 14, 6]
```

8 and 10 are in order

10 and 13 are in order

13 and 14 are in order

6 and 14 should be swapped:

```
[ 4, 1, 8, 10, 13, 6, 14]
```

Bubble Sort Example (Cont)

```
[ 4, 1, 8, 10, 13, 6, 14]
```

<u>Second pass:</u>

4 and 1 should be swapped:

```
[ 1, 4, 8, 10, 13, 6, 14]
```

4 and 8 are in order

8 and 10 are in order

10 and 13 are in order

13 and 6 should be swapped:

```
[ 1, 4, 8, 10, 6, 13, 14]
```

13 and 14 are in order

Bubble Sort Example (Cont)

```
[ 1, 4, 8, 10, 6, 13, 14]
```

<u>Third pass:</u>
10 and 6 should be swapped:

```
[ 1, 4, 8, 6, 10, 13, 14]
```

<u>Fourth pass:</u>
8 and 6 should be swapped:

```
[ 1, 4, 6, 8, 10, 13, 14]
```

Bubble Sort Run Time

- For a list of size **N**, how much work do we do for a single pass?
  - **N**
- How many passes will we have to do?
  - **N**
- What is the run time of Bubble Sort?
  - $\mathbf{N}^2$

Bubble Sort Video

Video from https://www.youtube.com/watch?v=lyZQPjUT5B4

Quicksort

Quicksort Algorithm

- Here’s another method:

1. Start with the number on the far right
2. Put everything less than that number on the left of it and everything greater than it on the right of it
3. Quicksort the left side and the right side

- Does this method remind you of anything?

Quicksort Run Time

- For a list of size **`N`**, how many steps does it take to move everything less than the last number to the left and everything greater than the last number to the right?

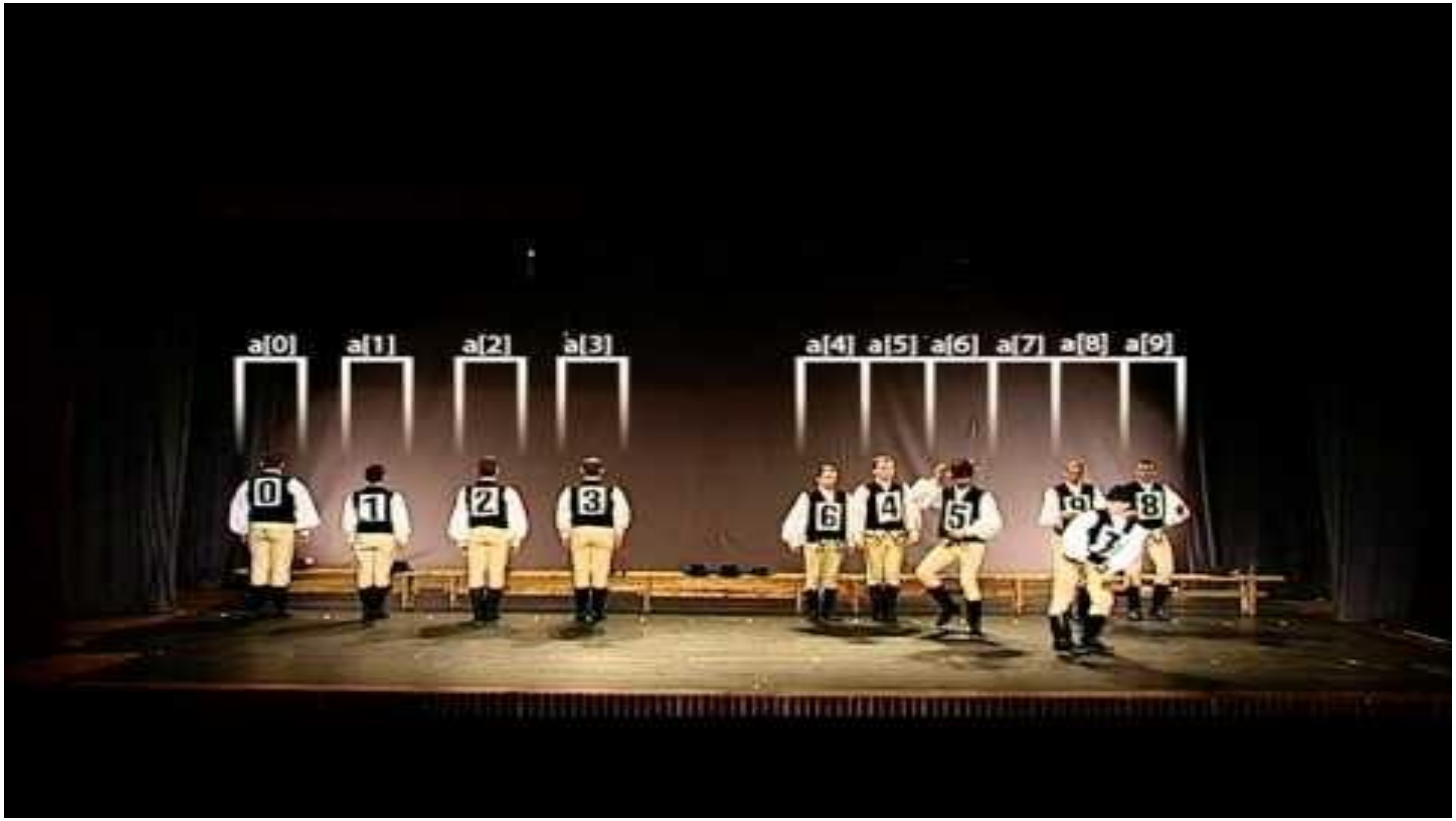
- **`N`**

Quicksort Run Time

- How many times will the algorithm divide the list in half?
- **`lg(N)`**

- What is the run time of Quicksort?
- **`N * lg(N)`**

Quicksort Video



Video from https://www.youtube.com/watch?v=ywWBy6J5gz8

Announcements

- Final is Thursday, December 15th (3:30 – 5:30)
- “Hour of code” volunteer opportunity
  - http://www.signupgenius.com/go/10c0f45aaab2aabfc1-hour
- Project 2 out now
  - Due on ***Tuesday***, December 13th
- Survey #3 also out – follow link in announcement