Recursive Sorting

Java

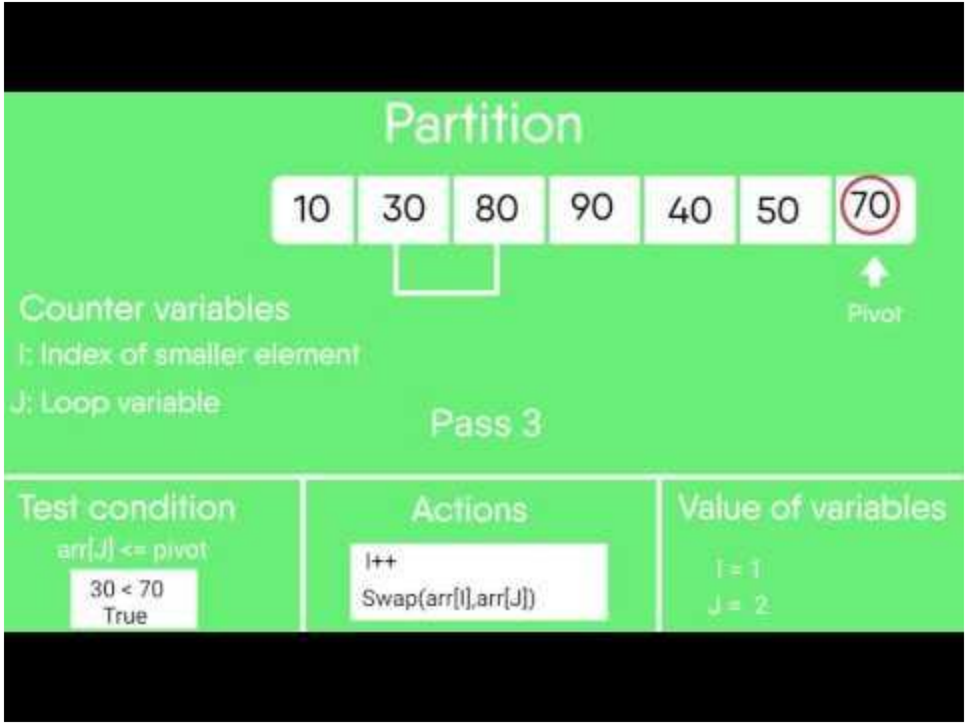
Mr. Poole

Big O of Sorting Algorithms

Algorithm	Time Complexity (Best)	Time Complexity (Average)	Time Complexity (Worst)	Space Complexity
Bubble Sort	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$
Insertion Sort	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$

Sorting - Quicksort

Sorting - Quick Sort



What is the base case?

```
void quickSort(int arr[], int low, int high)
{
    if (low < high)
    {
        /* pi is partitioning index, arr[p] is now
        at right place */
        int pi = partition(arr, low, high);

        // Separately sort elements before
        // partition and after partition
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}
```

```
int partition (int arr[], int low, int high)
{
    int pivot = arr[high]; // pivot
    int i = (low - 1); // Index of smaller element and indicat

    for (int j = low; j <= high - 1; j++)
    {
        // If current element is smaller than the pivot
        if (arr[j] < pivot)
        {
            i++; // increment index of smaller element
            swap(&arr[i], &arr[j]);
        }
    }
    swap(&arr[i + 1], &arr[high]);
    return (i + 1);
}
```

What is the base case?

```
void quickSort(int arr[], int low, int high)
{
    if (low < high)
    {
        /* pi is partitioning index, arr[p] is now
        at right place */
        int pi = partition(arr, low, high);

        // Separately sort elements before
        // partition and after partition
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}
```

if (low > high)

Basically if the index of the array from the low side every goes past the index of the high side.
Low side adds 1, while high side subtracts 1

```
int partition (int arr[], int low, int high)
{
    int pivot = arr[high]; // pivot
    int i = (low - 1); // Index of smaller element and indicat

    for (int j = low; j <= high - 1; j++)
    {
        // If current element is smaller than the pivot
        if (arr[j] < pivot)
        {
            i++; // increment index of smaller element
            swap(&arr[i], &arr[j]);
        }
    }
    swap(&arr[i + 1], &arr[high]);
    return (i + 1);
}
```

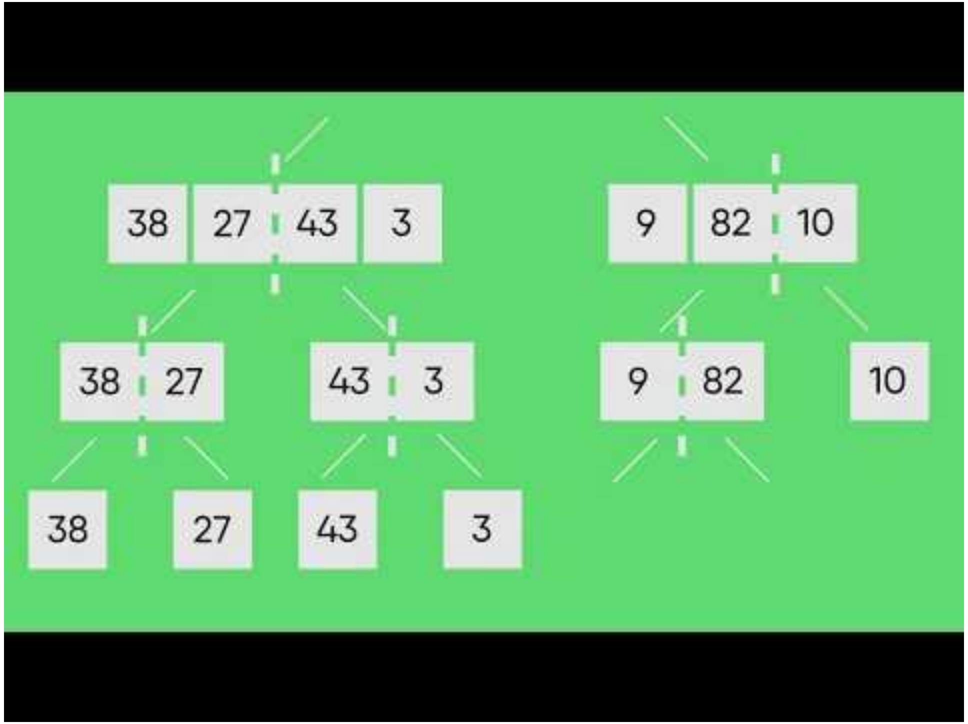
What is the runtime of QuickSort?

Big O of Quicksort

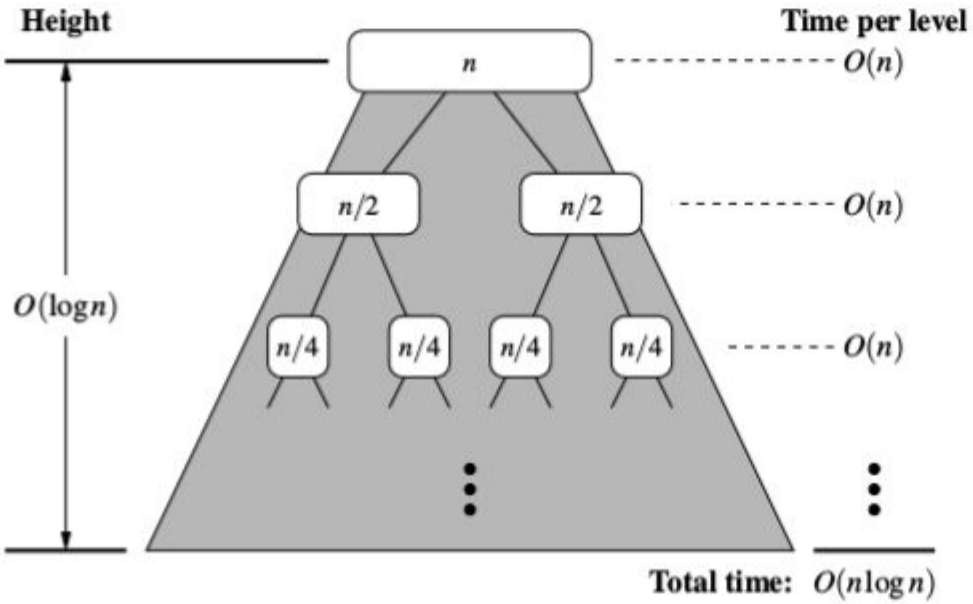
Time Complexity (Best)	Time Complexity (Average)	Time Complexity (Worst)	Space Complexity
O(n log n)	O(n log n)	O(n^2)	O(log n)

Sorting - Merge Sort

Sorting - Merge Sort



Runtime of MergeSort



Examples of Runtime

Here are the running times of some operations we might perform on the phone book, from fastest to slowest:

- **O(1) (in the worst case):** Given the page that a business's name is on and the business name, find the phone number.
- **O(1) (in the average case):** Given the page that a person's name is on and their name, find the phone number.
- **O(log n):** Given a person's name, find the phone number by picking a random point about halfway through the part of the book you haven't searched yet, then checking to see whether the person's name is at that point. Then repeat the process about halfway through the part of the book where the person's name lies. (This is a binary search for a person's name.)
- **O(n):** Find all people whose phone numbers contain the digit "5".
- **O(n):** Given a phone number, find the person or business with that number.
- **O(n log n):** There was a mix-up at the printer's office, and our phone book had all its pages inserted in a random order. Fix the ordering so that it's correct by looking at the first name on each page and then putting that page in the appropriate spot in a new, empty phone book.

Examples of Runtime

For the below examples, we're now at the printer's office. Phone books are waiting to be mailed to each resident or business, and there's a sticker on each phone book identifying where it should be mailed to. Every person or business gets one phone book.

- **O(n log n):** We want to personalize the phone book, so we're going to find each person or business's name in their designated copy, then circle their name in the book and write a short thank-you note for their patronage.
- **O(n^2):** A mistake occurred at the office, and every entry in each of the phone books has an extra "0" at the end of the phone number. Take some white-out and remove each zero.
- **O(n · n!):** We're ready to load the phonebooks onto the shipping dock. Unfortunately, the robot that was supposed to load the books has gone haywire: it's putting the books onto the truck in a random order! Even worse, it loads all the books onto the truck, then checks to see if they're in the right order, and if not, it unloads them and starts over. (This is the dreaded **bogo sort**.)
- **O(n^n):** You fix the robot so that it's loading things correctly. The next day, one of your co-workers plays a prank on you and wires the loading dock robot to the automated printing systems. Every time the robot goes to load an original book, the factory printer makes a duplicate run of all the phonebooks! Fortunately, the robot's bug-detection systems are sophisticated enough that the robot doesn't try printing even more copies when it encounters a duplicate book for loading, but it still has to load every original and duplicate book that's been printed.

Link to thread

Examples of Runtime

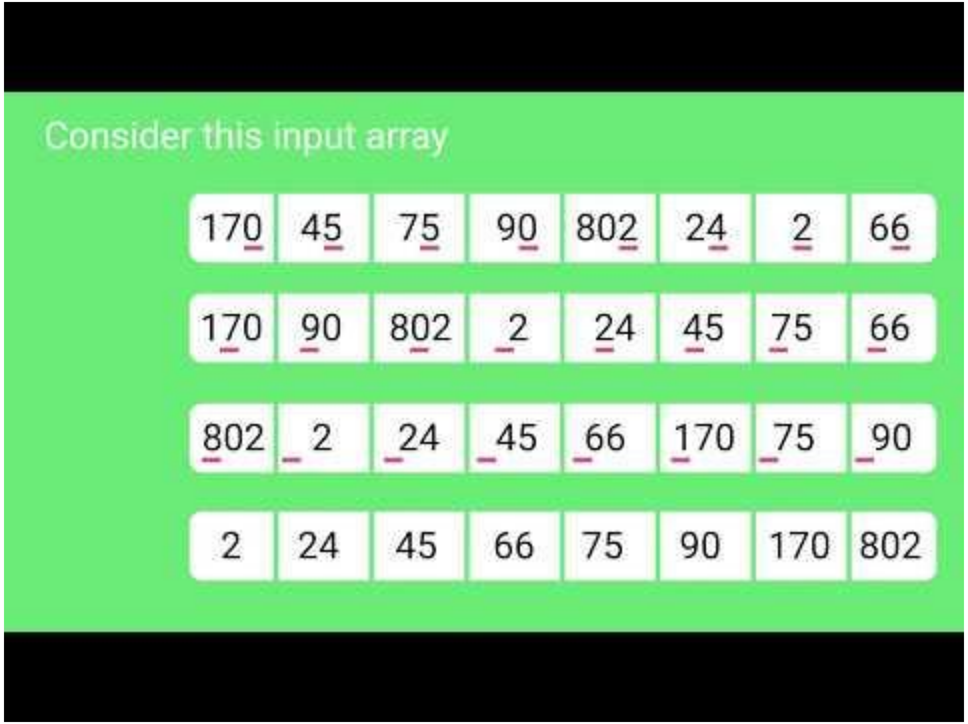
`O(log N)` basically means time goes up linearly while the `n` goes up exponentially. So if it takes `1` second to compute `10` elements, it will take `2` seconds to compute `100` elements, `3` seconds to compute `1000` elements, and so on.

It is `O(log n)` when we do divide and conquer type of algorithms e.g binary search. Another example is quick sort where each time we divide the array into two parts and each time it takes `O(N)` time to find a pivot element. Hence it `N O(log N)`

[Link to thread]

Sorting - Radix Sort

Sorting - Radix Sort



What is Radix Sort runtime?

```
// initializing the hash array with null to all
for (int i = 0; i < 10; i++)
    bins[i] = NULL;

// first loop working for a constant time only and inner
// loop is iterating through the array to store elements
// of array in the linked list by their digits value
for (int i = 0; i < d; i++) {
    for (int j = 0; j < sz; j++) // bins updation
        insert(bins[(arr[j] / (int)pow(10, i)) % 10],
               arr[j]);

    int x = 0, y = 0;
    // write back to the array after each pass

    while (x < 10) {
        while (bins[x] != NULL)
            arr[y++] = del(bins[x]);
        x++;
    }
}
```

RADIX SORT BIG O

Time Complexity (Best)	**Time Complexity (Average)**	**Time Complexity (Worst)**	**Space Complexity**
$O(nk)$	$O(nk)$	$O(nk)$	$O(n + k)$

- *n* - length of array
- *k* - number of digits(average)

How to do Recursion

Recursion Steps - MUST HAVE

1. Base Case (i.e. when to stop)

2. Work towards Base Case

3. Recursive Call (i.e. calling itself)

For example, we can define the operation "find your way home" as:

1. If you are at home, stop moving.

2. Take one step toward home.

3. "find your way home".

Let’s try out Factorial!

```
int factorial(int n)
{
   return n * factorial(n - 1);
}
```

Factorial with a Base Case

```
int factorial(int n)
{
    if(n == 1)
    {
        return 1;
    }
    else
    {
        return n * factorial(n - 1);
    }
}
```

Let’s try recursion!

Given one number, print out that number all the way to 1 and back up to the number.

Ex: function(3)

Output: 3 2 1 1 2 3

Example

Next Lecture:
Lists, Stacks, and Queues