

CS 32: Discussion 1D

TA: Shichang Zhang

LA: Stephanie Doan, Rish Jain

Midterm

Any comments on it?

Note: We do not know when it will be graded unfortunately

Announcements

- LA Workshop on Big O and Algorithm Analysis(May 19th 7PM-8PM)
- Project 3 due 11 pm Tuesday (May 18th)
- Homework 4 due 11 pm Tuesday (May 25th)

Overview

- Big O
- Sorting
- Worksheet

Algorithm Efficiency

- Quantify the efficiency of a program.
- The magnitude of time and space cost for an algorithm given certain size of input.
 - Time complexity: quantifies the run time.
 - Space complexity: quantifies the usage of the memory (or sometimes hard disk drives, cloud disk drives, etc.).
- Naturally, the size of input determines how long a program runs.
 - Often, the larger the size of input, the longer the run time. But not always that case.
 - Consider: sort an array of 1,000 items and 1,000,000 items vs get size of an array of 1,000 items and 1,000,000 items
- Big-O notation

Big-O Notation

Basically it says how many operations you are doing given the input size n .

By convention, we ignore all the constants and the lower order terms.

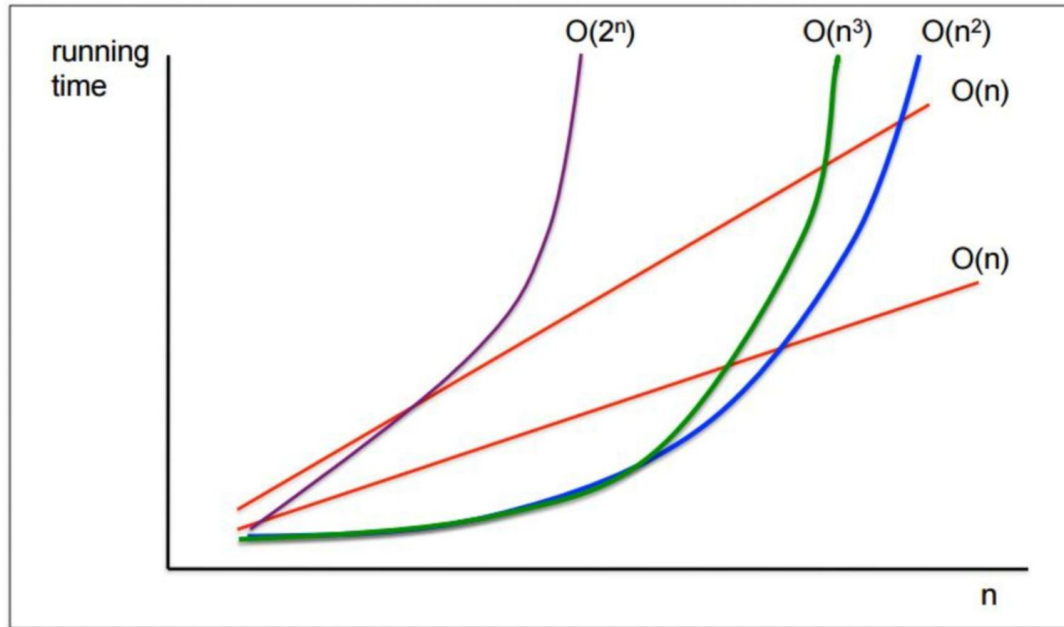
No need to memorize definitions. Example: if your program takes,

- about n steps $\rightarrow O(n)$
- about $2n$ steps $\rightarrow O(n)$
- about n^2 steps $\rightarrow O(n^2)$
- about $3n^2+10n$ steps $\rightarrow O(n^2)$
- about 2^n steps $\rightarrow O(2^n)$

Question: What is the speed of growth for typical function?

$f(n) = \log(n) / n / n^2 / 2^n / n!$

Big-O Notation



Big-O Arithmetic

Generally,

- If things happen sequentially, we add Big-Os;
- If one thing happen within another, then we multiply Big-Os.
- Simple rule: Watch the **LOOPS** in your programs!

Rules:

$$O(f(n)) + O(g(n)) = O(\max(f(n), g(n)))$$

$$O(f(n)) \times O(g(n)) = O(f(n) \times g(n))$$

Big-O Arithmetic Example

Q: What is following numbers in Big-O notation?

Q1: $n + 3 \cdot n \log(n)$

Q2: $\exp(n) + n^3 + \log(n)$

Q3: $n^5 + 5 \cdot n$

Big-O Arithmetic Example

Q: What is following numbers in Big-O notation?

Q1: $n + 3 \cdot n \log(n)$

A1: $O(n \log(n))$

Q2: $\exp(n) + n^3 + \log(n)$

A2: $O(\exp(n))$

Q3: $n^5 + 5 \cdot n$

A3: $O(n^5)$

Big-O Arithmetic Example

Q: What is result in Big-O notation?

Q1: $O(n^2 + n\log(n)) + O(n)$

Q2: $O(n^2 + n\log(n)) * O(n)$

Big-O Arithmetic Example

Q: What is result in Big-O notation?

Q1: $O(n^2 + n \log(n)) + O(n)$

A1: $O(n^2)$

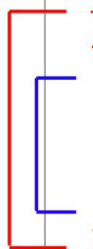
Q2: $O(n^2 + n \log(n)) * O(n)$

A2: $O(n^3)$

Ex. Code Analysis

- Task: Find all pairs from one array (Note: [1,2] and [2,1] are considered different pairs)

```
int all_pairs(array arr, size n, value v)
{
    for (int i=0; i<n; i++)
    {
        for (int j=0; i<n; j++)
        {
            if (i != j)
                cout << "Pair:" << arr[i] << "and" << arr[j] << endl;
        }
    }
    return -1;
}
```

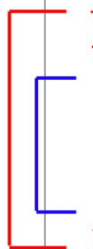
A diagram consisting of two nested brackets on the left side of the code. The outer bracket is red and spans the entire function body from the first 'for' loop to the 'return' statement. The inner bracket is blue and spans the inner 'for' loop and its associated 'if' statement and 'cout' line.

Q: What is the time complexity of this piece of code in Big-O notation?

Ex. Code Analysis

- Task: Find all pairs from one array (Note: [1,2] and [2,1] are considered different pairs)

```
int all_pairs(array arr, size n, value v)
{
    for (int i=0; i<n; i++)
    {
        for (int j=0; j<n; j++)
        {
            if (i != j)
                cout << "Pair:" << arr[i] << "and" << arr[j] << endl;
        }
    }
    return -1;
}
```

A diagram consisting of two nested brackets on the left side of the code. The outer bracket is red and spans the entire function body from the first 'for' loop to the 'return' statement. The inner bracket is blue and spans the inner 'for' loop and its 'if' statement block.

Q: What is the time complexity of this piece of code in Big-O notation?

A: $O(n^2)$

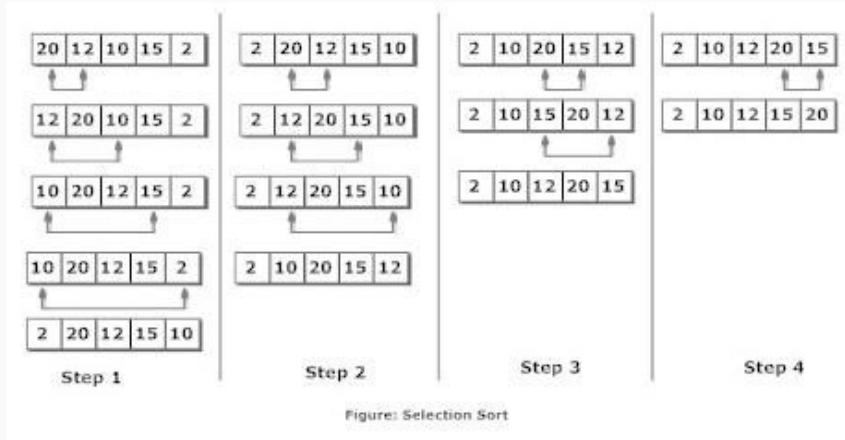
Sorting

Five main sorting algorithms

- Selection Sort
- Insertion Sort
- Bubble Sort
- MergeSort
- QuickSort

Selection Sort

Repeatedly find minimum element in subarray and place at beginning of subarray by swapping out the first element of that subarray.



5 3 4 1 2

Selection Sort

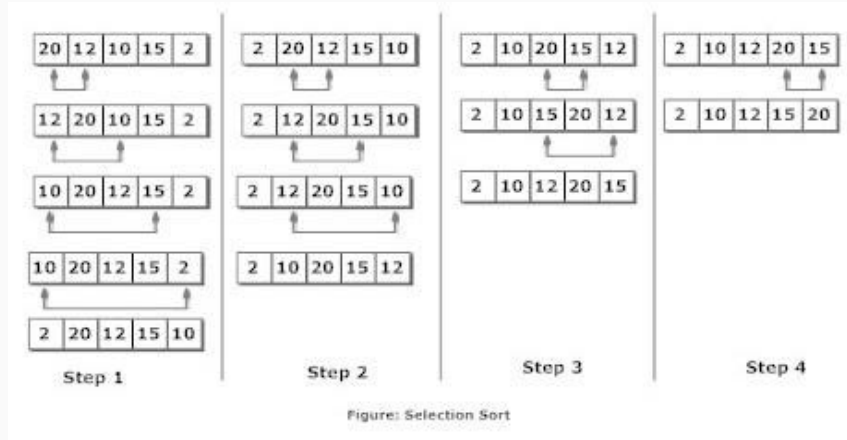
	8
	5
	2
	6
	9
	3
	1
	4
	0
	7

What is the time complexity of this sort:

- Best case: ??
- Worst case: ??

Selection Sort

Repeatedly find minimum element in subarray and place at beginning of subarray by swapping out the first element of that subarray.



5 3 4 1 2

Selection Sort

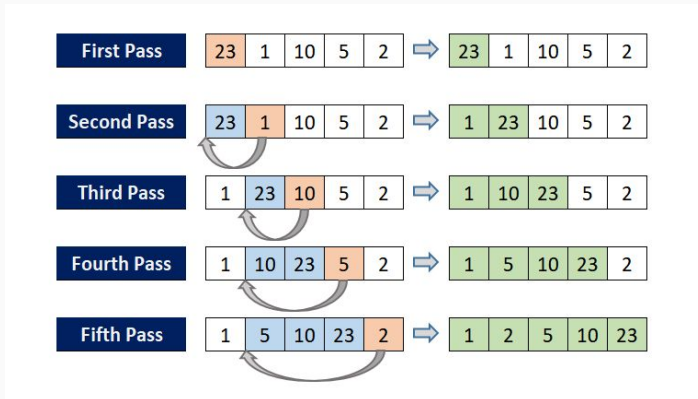
	8
	5
	2
	6
	9
	3
	1
	4
	0
	7

What is the time complexity of this sort:

- Best case: $O(n^2)$
- Worst case: $O(n^2)$

Insertion Sort

Builds final sorted array one subarray at a time, which grows from one item to the whole sorted array.



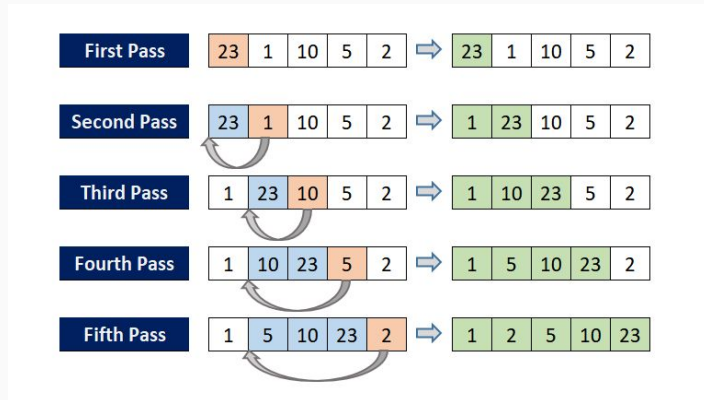
6 5 3 1 8 7 2 4

What is the time complexity of this sort:

- Best case: ??
- Worst case: ??

Insertion Sort

Builds final sorted array one subarray at a time, which grows from one item to the whole sorted array.



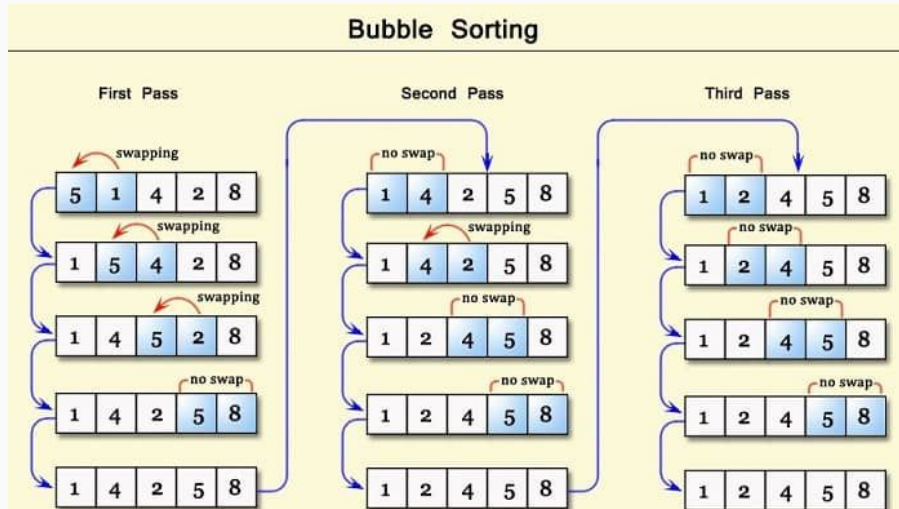
6 5 3 1 8 7 2 4

What is the time complexity of this sort:

- Best case: $O(n)$ when array is already sorted
- Worst case: $O(n^2)$ when array is in reverse order

Bubble Sort

- Repeatedly swap adjacent elements if they're in the wrong order



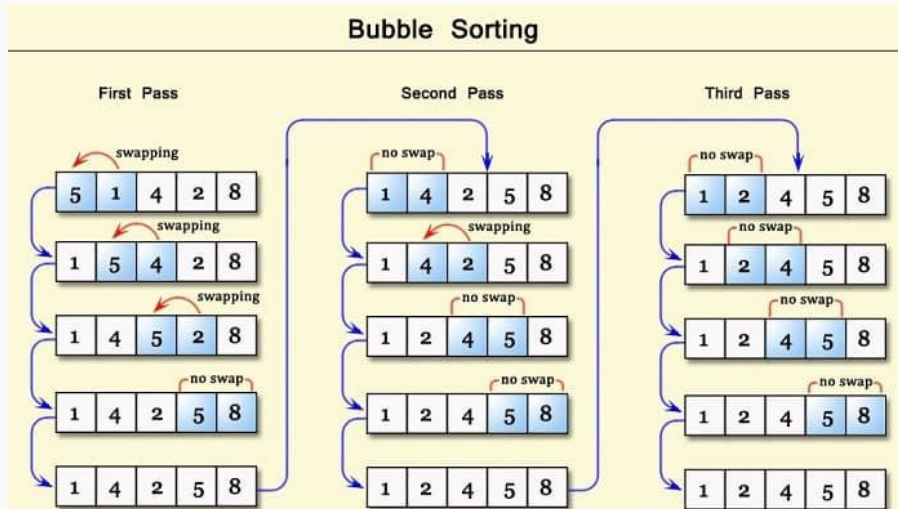
6 5 3 1 8 7 2 4

What is the time complexity of this sort:

- Best case? ??
- Worst case? ??

Bubble Sort

- Repeatedly swap adjacent elements if they're in the wrong order



6 5 3 1 8 7 2 4

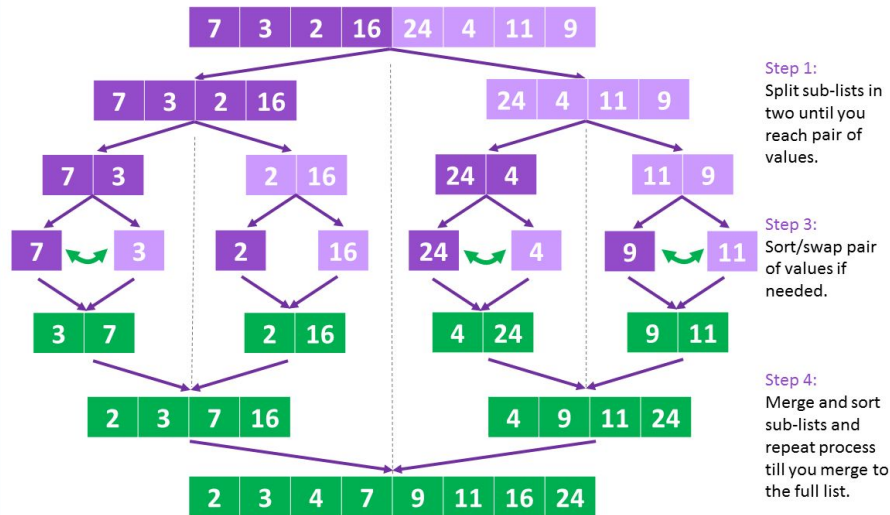
What is the time complexity of this sort:

- Best case? $O(n)$ if array is already sorted
- Worst case? $O(n^2)$ if array is reversed

Merge Sort

Divide unsorted list into n sublists each of 1 element (since each list of one element is sorted) and merge sublists to produce new sorted sublists until we get one list of original size.

Merge Sort



6 5 3 1 8 7 2 4

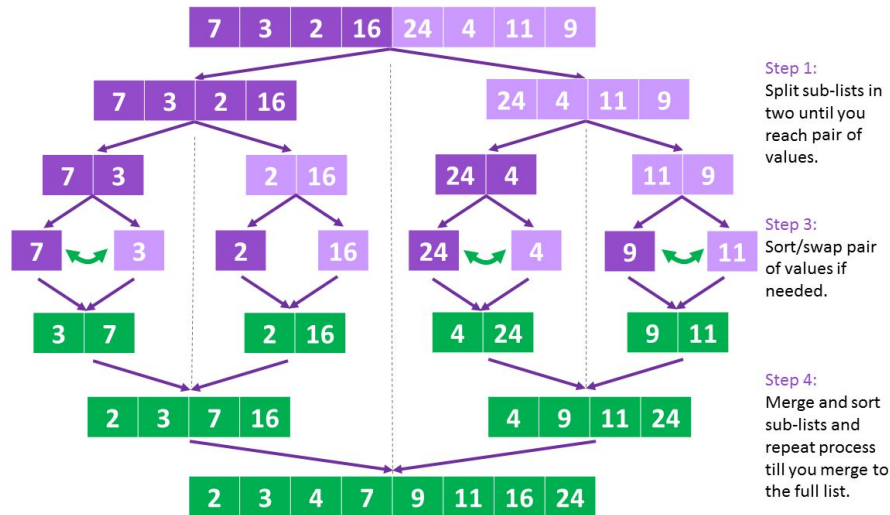
What is the time complexity of this sort:

- Best case: ??
- Worst case: ??

Merge Sort

Divide unsorted list into n sublists each of 1 element (since each list of one element is sorted) and merge sublists to produce new sorted sublists until we get one list of original size.

Merge Sort



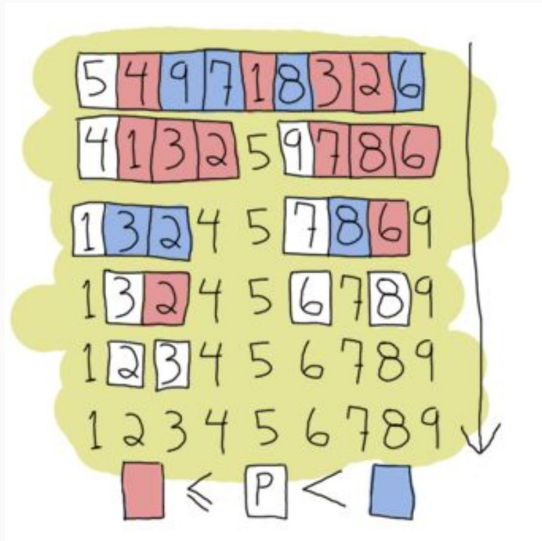
6 5 3 1 8 7 2 4

What is the time complexity of this sort:

- Best case: $O(n \cdot \log(n))$
- Worst case: $O(n \cdot \log(n))$

QuickSort

Set a pivot. Move the small (large) numbers to the left (right) of the pivot. Repeat it recursively for the left and right part.

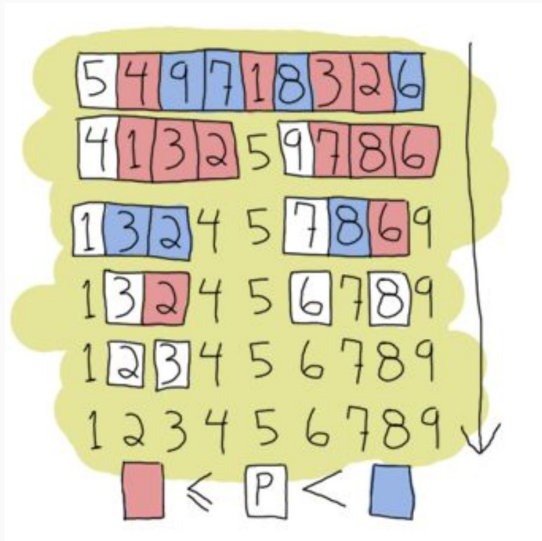


What is the time complexity of this sort:

- Best case: ??
- Worst case: ??

QuickSort

Set a pivot. Move the small (large) numbers to the left (right) of the pivot. Repeat it recursively for the left and right part.



What is the time complexity of this sort:

- Best case: $O(n \log(n))$
- Worst case: $O(n^2)$ when the pivot you pick is always the smallest(largest) one.

Break: 5 mins

Worksheet

Worksheet Problem Number 8 Visual

Since codeshare doesn't allow us to bold or underline. Here is a better image

3 7 4 9 5 2 6 1

3 7 4 9 5 2 6 1

3 **7** 4 9 5 2 6 1

3 **4** 7 9 5 2 6 1

3 4 7 **9** 5 2 6 1

3 4 **5** 7 9 2 6 1

2 3 4 5 7 9 6 1

2 3 4 5 **6** 7 9 1

1 2 3 4 5 6 7 9

Codeshare

Room 1

Room 2

Room 3

Room 4

Worksheet Solution