

CSE12 - Lecture 14

Wednesday, November 1, 2023 8:00 AM

PA5 released today

PA4 & PA2 Late/Resubmit → slip day ^{until} tomorrow @ 8am

Lecture 14

Quicksort: Another magical (recursive) algorithm

<https://www.youtube.com/watch?v=ywWBy6J5gz8>

14	4	9	12	15	8	19	2
----	---	---	----	----	---	----	---

Select a **pivot** element:

14	4	9	12	15	8	19	2
----	---	---	----	----	---	----	---

"Partition" the elements in the array (smaller or equal to pivot, larger or equal to pivot)

2	4	9	8	15	12	19	14
---	---	---	---	----	----	----	----

Magically sort the smaller elements and the larger elements (Quicksort)

2	4	8	9	12	15	19	21
---	---	---	---	----	----	----	----

Quick Sort: Using a "good" pivot

How many levels will there be if you choose a pivot that divides the list in half?

- A. 1
- B. $\log(n)$
- C. n
- D. $n \cdot \log(n)$
- E. n^2

If the time to partition on each level takes N comparisons, how long does Quicksort take with a good partition?

- A. $O(1)$
- B. $O(\log(n))$
- C. $O(n)$
- D. $O(n \cdot \log(n))$
- E. $O(n^2)$

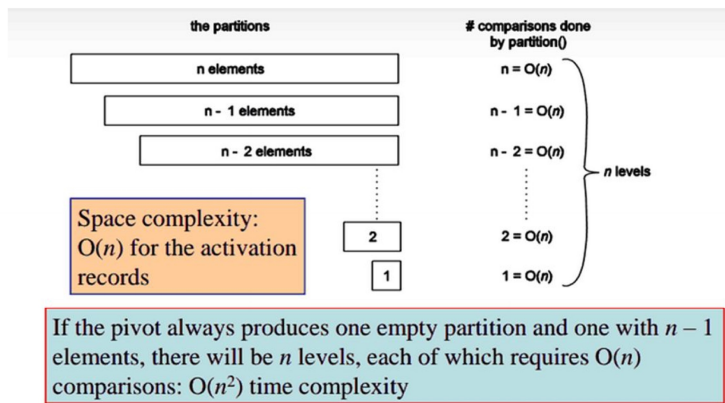
Which of these choices would be the worst choice for the pivot?

- A. The minimum element in the list
- B. The last element in the list
- C. The first element in the list
- D. A random element in the list

best case
→ pivot → median value

Name: _____ PID: _____ Code: 1790

Quick sort with a bad pivot



Which of these choices is a better choice for the pivot?

- A. The first element in the list
- ☒ B. A random element in the list
- C. They are about the same

There are many ways to partition!

Quick sort - Middle Pivot

- We always pick the middle location as pivot
- The data we sort is {2, 3, 1, 5, 4, 6, 7}

After the first split, what is the order of elements in the list that was \leq pivot?

- A. 1 2 3 4
- ☒ B. 2 3 1 4
- C. 4 3 2 1
- D. 3 4 1 2
- E. None of the above

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

What is the first pivot? 15 P index 4

How would you sort the data around the pivot?

lowIndex = low; 0
highIndex = high; 8-1=7
pivotIndex = (high-low)/2 = (8-0)/2 = 4

pivotValue = value [pivotIndex]

update if it moves

return pivotIndex

low = 0, high = 8
12 2 9 4 3 8 15 19
↑ ↑
H L

loop

check left of middle

if value [lowIndex] < pivotValue
lowIndex++

else
swap (lowIndex, highIndex);
highIndex--

check right of middle

if value [highIndex] > pivotValue
highIndex--

else
swap (lowIndex, highIndex);
lowIndex++

if (low > high)
return / break
done = true

```
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;
        }
    }
    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;
                }
            }
        }
    }
}
```

sorted array
1 2 3 4 5
 $O(n)$

```
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;
        }
    }

    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);
    }
}
```

```

    qsort(array, splitAt + 1, high);
}
public static void sort(String[] array) {
    qsort(array, 0, array.length);
}
}

```

	Insertion	Selection	Merge	Quick
Best case time	Sorted array $O(N)$	$O(N^2)$	$O(N * \log_2(N))$	median value $O(N * \log_2(N))$
Worst case time	Reverse sorted array $O(N^2)$	$O(N^2)$	$O(N * \log_2(N))$	$O(N^2)$ Average case: $O(N * \log_2(N))$
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)

Last note about sorting

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	$O(\log n)$	No

? "l", "Grg"
? "l", "Elsa"

"Grg", "Elsa"
"Elsa", "Grg"
→ unstable ordering

Java Array Int sorting
↳ QuickSort → primitives
↳ MergeSort → Objects