# CS122 Algorithms and Data Structures

MW 11:00 am - 12:15 pm, MSEC 101

Instructor: Xiao Qin

Lecture 20: Sorting (2)



## Elementary Sorting Algorithms

- Selection Sort, Insertion Sort, and Bubble Sort all have a worst-case time of $O(n^2)$, making them impractical for large arrays.
- But they are easy to program, easy to debug.
- Insertion Sort also has good performance when the array is nearly sorted to begin with.
- But more sophisticated sorting algorithms are needed when good performance is needed in all cases for large arrays.



## Sorting Algorithms and Average Case Number of Comparisons

**Simple Sorts**
- **Insertion Sort**
- **Selection Sort**
- **Bubble Sort**

**$O(N^2)$**

**More Complex Sorts**
- **Heap Sort**
- **Quick Sort**
- **Merge Sort**

**O(N*log N)**



## Heaps

- A heap has two properties.
- Heaps can be implemented by arrays.
- Convert an array into a heap.
- A top down method
- A bottom up method



## Convert an Array into a Heap

- The Floyd Algorithm (bottom-up)

```
FloydAlgorithm(heap[])
   for (i = index of last nonleaf; i>=0; i--) do
      restore the heap property for the tree whose root is
      heap[i] by calling movedown(heap, i,n-1);
   endfor
```

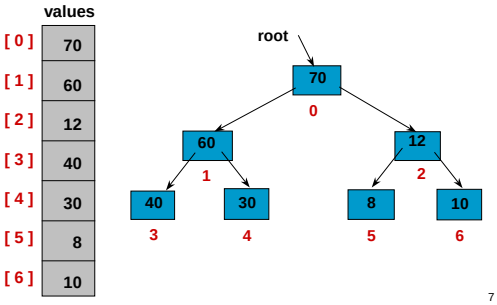


## Heap Sort Approach

**First, make the unsorted array into a heap by satisfying the order property. Then repeat the steps below until there are no more unsorted elements.**

- **Take the root (maximum) element off the heap by swapping it into its correct place in the array at the end of the unsorted elements.**
- **Reheap the remaining unsorted elements. (This puts the next-largest element into the root position).**

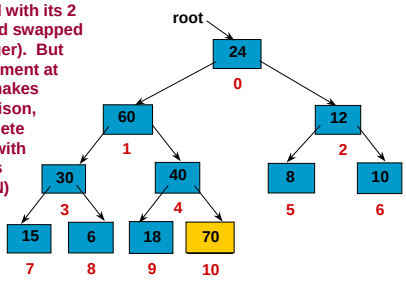
## Example:
### After creating the original heap




7

## Heap Sort:
## How many comparisons?

In reheap down, an element is compared with its 2 children (and swapped with the larger). But only one element at each level makes this comparison, and a complete binary tree with N nodes has only $O(\log_2 N)$ levels.




8

## Quicksort Algorithm

Given an array of *n* elements (e.g., integers):

- If array only contains one element, return
- Else
  - pick one element to use as *pivot.*
  - Partition elements into two sub-arrays:
    - Elements less than or equal to pivot
    - Elements greater than pivot
  - Quicksort two sub-arrays
  - Return results

9

## Example

We are given array of n integers to sort:

| 40 | 20 | 10 | 80 | 60 | 50 | 7 | 30 | 100 |
|---|---|---|---|---|---|---|---|---|

10

## Pick Pivot Element

There are a number of ways to pick the pivot element. In this example, we will use the first element in the array:

| 40 | 20 | 10 | 80 | 60 | 50 | 7 | 30 | 100 |
|---|---|---|---|---|---|---|---|---|

11

## Partitioning Array

Given a pivot, partition the elements of the array such that the resulting array consists of:

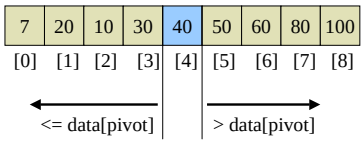
1. One sub-array that contains elements >= pivot
2. Another sub-array that contains elements < pivot

The sub-arrays are stored in the original data array.

Partitioning loops through, swapping elements below/above pivot.

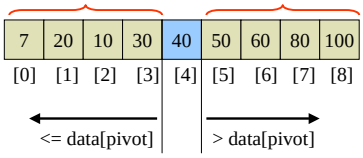
12

## Partition Result




13

## Recursion: Quicksort Sub-arrays




14

## Quicksort Analysis

- Assume that keys are random, uniformly distributed.
- What is best case running time?
  - Recursion:
    1. Partition splits array in two sub-arrays of size n/2
    2. Quicksort each sub-array
  - Depth of recursion tree? $O(\log_2 n)$
  - Number of accesses in partition? O(n)

15

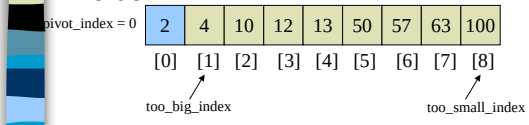
## Quicksort Analysis

- Assume that keys are random, uniformly distributed.
- Best case running time: $O(n \log_2 n)$
- Worst case running time?

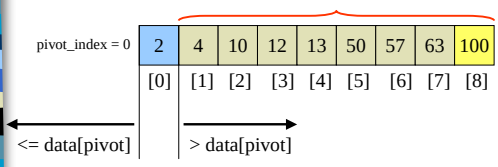
16

## Quicksort: Worst Case

- Assume first element is chosen as pivot.
- Assume we get array that is already in order:




17




18

## Quicksort Analysis

- Assume that keys are random, uniformly distributed.
- Best case running time: $O(n \log_2 n)$
- Worst case running time?
  - Recursion:
    1. Partition splits array in two sub-arrays:
       - one sub-array of size 0
       - the other sub-array of size n-1
    2. Quicksort each sub-array
  - Depth of recursion tree? O(n)
  - Number of accesses per partition? O(n)

19

## Quicksort Analysis

- Assume that keys are random, uniformly distributed.
- Best case running time: $O(n \log_2 n)$
- Worst case running time: $O(n^2)$!!!
- What can we do to avoid worst case?

20