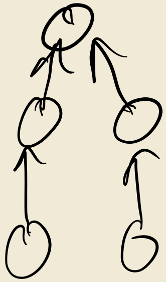
sorting

slides
bit.ly/abhi-disc

attendance
bit.ly/abhi-attendance

announcements

1. Homework 7 due Tuesday 4/12
2. Week 12 Survey due Tuesday 4/12
3. Project 3 ~~coming up!!~~! Released
4. Lab 13 due Friday 4/15



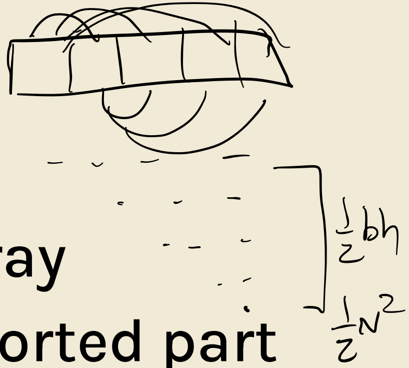
general questions, lecture, etc.

selection sort

1) start with the first element of the array
2) find the smallest element in the unsorted part of the array
3) swap this ^ with the current element we’re at
4) move to the next element

runtime:

selection sort



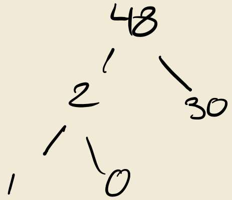
1) start with the first element of the array
2) find the smallest element in the unsorted part of the array
3) swap this ^ with the current element we're at
4) move to the next element

runtime: O(N^2)

demo

selection sort

heapsort



1) “heapify” the array into a max heap
2) remove the max and add this node to the end of the output array
3) repeat step 2 a total of N times (where there are N items to be sorted)

runtime:

heapsort

1) "heapify" the array into a max heap
2) remove the max and add this node to the end of the output array
3) repeat step 2 a total of N times (where there are N items to be sorted)

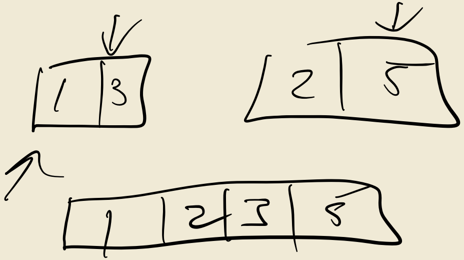
runtime: 1) O(NlogN) +

heapsort

1) "heapify" the array into a max heap
2) remove the max and add this node to the end of the output array
3) repeat step 2 a total of N times (where there are N items to be sorted)

runtime: 1) $O(N\log N)$ + 2-3) $O(N)$ = $O(N\log N)$

mergesort



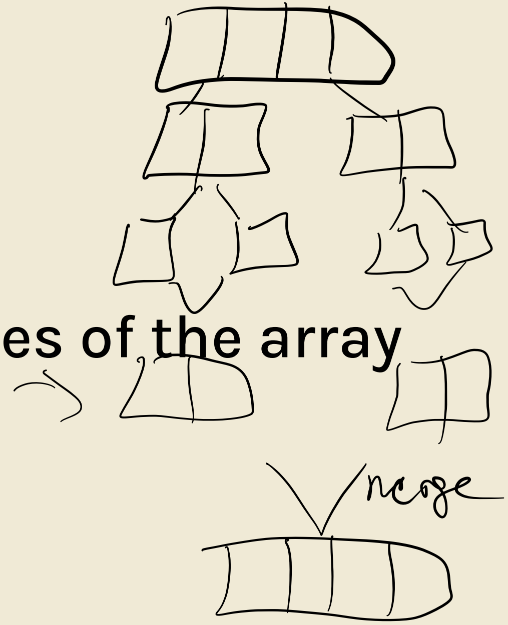
1) split the array roughly in two
2) recursively [merge]sort both halves of the array
3) "merge" the sorted halves

runtime:

mergesort

1) split the array roughly in two
2) recursively [merge]sort both halves of the array
3) “merge” the sorted halves

runtime: $\Theta(N \log N)$



demo

mergesort

1 5 2 3
1 2 5 3
1 2 3 5

insertion sort

1) start with a pointer at the leftmost element
2) keep swapping with the element to the left if the element to the left is greater than the current element
3) advance the pointer to the element to the right

runtime:

9 8 7 6

1 2 4 3

insertion sort

1) start with a pointer at the leftmost element
2) keep swapping with the element to the left if the element to the left is greater than the current element
3) advance the pointer to the element to the right

runtime: O(N^2) (but Θ(N) best case)

best for almost sorted/small arrays!

demo

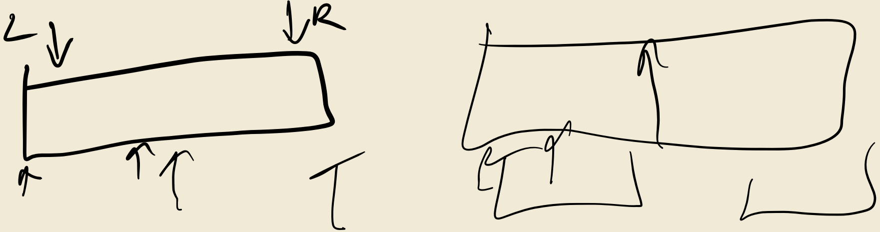
insertion sort

quicksort

1) “partition” array around the leftmost element (pivot)
    a) elements < pivot are put to the left of the pivot, elements >= pivot are put to the right
2) recursively quicksort the left and right subarrays

runtime:

quicksort



1) “partition” array around the leftmost element (pivot)
   a) elements < pivot are put to the left of the pivot, elements >= pivot are put to the right
2) recursively quicksort the left and right subarrays

runtime: Θ(NlogN) (expected), Θ(N^2) (worst case)

demo

quicksort

	Best Case Runtime	Worst Case Runtime	Space	Demo	Notes
Selection Sort	$\Theta(N^2)$	$\Theta(N^2)$	$\Theta(1)$	Link	
Heapsort (in place)	$\Theta(N)$*	$\Theta(N \log N)$	$\Theta(1)$	Link	Bad cache (61C) performance.
Mergesort	$\Theta(N \log N)$	$\Theta(N \log N)$	$\Theta(N)$	Link	Fastest of these ^
Insertion Sort (in place)	$\Theta(N)$	$\Theta(N^2)$	$\Theta(1)$	Link	Best for small N or almost sorted.
QuickSort LTHS (left pivote tony hoare, shuffled)	$\Theta(N \log N)$	$\Theta(N^2)$	$\Theta(\log N)$ (call stack)	Link	Empirically the fastest sort, rare worst case
LSD Radix Sort	$\Theta(WN+WR)$	$\Theta(WN+WR)$	$\Theta(N+R)$	—	Alphabetical only
MSD Radix Sort	$\Theta(N+R)$	$\Theta(WN+WR)$	$\Theta(N+WR)$	—	Bad caching (61C)

*: An array of all duplicates yields linear runtime for heapsort.
N: Number of keys. R: Size of alphabet. W: Width of longest key.

(Josh Hug 2021)

worksheet

(on 61B website)

attendance

bit.ly/abhi-attendance

feedback

bit.ly/abhi-feedback

slides: **bit.ly/abhi-disc**