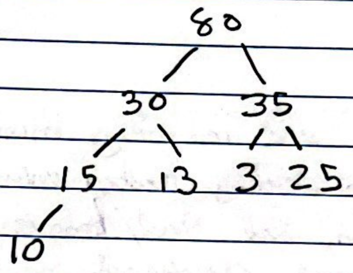
Emmet Blassingame

CS280: HW6

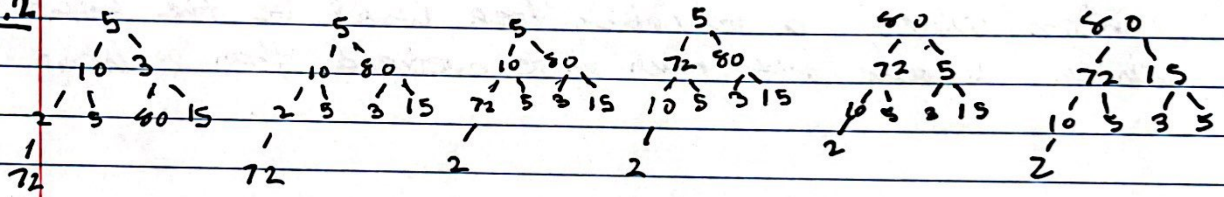
1

1.1



80	30	35	15	13	3	25	10

1.2



[5, 10, 3, 2, 5, 80, 15, 72] ← Original Array
 0 1 2 3 4 5 6 7

[5, 10, 80, 2, 5, 3, 15, 72] switch i=2 and i=5

[5, 10, 80, 72, 5, 3, 15, 2] switch i=4 and i=7

[5, 72, 80, 10, 5, 3, 15, 2] switch i=1 and i=3

[80, 72, 5, 10, 5, 3, 15, 2] switch i=0 and i=2

[80, 72, 15, 10, 5, 3, 5, 2] switch i=2 and i=6

↑ Final Array

CS280: HW6

1.3

Heaps are very fast for inserting values and retrieving minimum values. Yet, they don't support searching or deleting random values efficiently. If all you need is insertion, and find/remove min., a heap would be the best choice because overhead is lower and runtime faster. If you need to insert/remove/lookup random values, a red/black tree would be the better choice, although with much more overhead than a heap.

CS280: HW6

2.1

4, 22, 98, 1, 18, 15, 3, 91, 72, 5, 9 34, 2, 17, 46, 55 ← original array

$\boxed{4\ 22\ 98\ 1\ 18\ 15\ 3\ 91}\ \boxed{72\ 5\ 9\ 34\ 2\ 17\ 46\ 55}$ ← Divide into ~~equls~~ equal halves

4 22 98 1 18 15 3 91 72 5 9 34 2 17 46 55 ← continue Dividing until single elements

4 22 98 1 18 15 3 91 72 5 9 34 2 17 46 55

4 22 98 1 18 15 3 91 72 5 9 34 2 17 46 55

4 22 1 98 15 18 3 91 5 72 9 34 2 17 46 55 ← combine and sort

1 4 22 98 3 15 18 91 5 9 34 72 2 17 46 55 ← sort and compare

1 3, 4, 15, 18, 22, 91, 98 2, 5, 9, 17, 34, 46, 55, 72 ← sort in 2 halves

✓ 1, 2, 3, 4, 5, 9, 15, 17, 18, 22, 34, 46, 55, 72, 91, 98 ← Finished Array

↑ Final, sorted array

2.2 • Quicksort could take $O(n^2)$ time in a worst-case scenario. For example, when the chosen pivot point is always an extreme (smallest or largest) element. This happens when input array is sorted and either first or last element is picked as pivot.

CS280: HW6

2.3

Insertion Sort
- Benefits include good performance when dealing with smaller lists.
  - In-place sorting alg. so space requirement is minimal.
- Disadvantages include not performing as well with larger lists.
  - Since n^2 steps required for every n element to be sorted, insertion does not deal well with large lists.

Mergesort
- Benefits include being quicker for larger ~~lists~~ lists, being slightly faster than heap sort in many cases, and having O(nLogN) as worst time complexity.
- Disadvantages include slower for smaller data sets and using twice the memory of heap sort.

Heapsort
- Pros: • Consistent performance
  • Efficient
  • Does not use recursion (easier to understand)
- Cons:
  • considered unstable, expensive, and not efficient when working with large data sets.

Quicksort

Benefits: Fast and efficient, better time complexity than others. Space complexity of O(logN) so excellent choice with limited space.

Disadvantages:
- unstable
- when pivot is extreme, bad performance.
- difficult to implement.

Scanned with CamScanner