CS220 Midsem Cheatsheet

Asymptotic Notation

f is $O(g)$ if g overbounds f

f is $\Omega(g)$ if g underbounds f

f is $\Theta(g)$ if g exactly bounds f

Limit Rule

Suppose that $L := \lim_{n\to\infty} f(n)/g(n)$ exists. Then

→ if $L = 0$ then f is $O(g)$

→ if $L > 0$ then f is $\Theta(g)$

→ if $L = \infty$ then f is $\Omega(g)$

More Asymptotic Notation Rules

Irrelevance of factors	If $c > 0$ is constant then cf is $\Theta(f)$
Transitivity	If f is $O(g)$ and g is $O(h)$ then f is $O(h)$
Sum Rule	If f_1 is $O(g_1)$ and f_2 is $O(g_2)$ the $f_1 + f_2$ is $O(\max\{g_1, g_2\})$
Product Rule	If f_1 is $O(g_1)$ and f_2 is $O(g_2)$ then $f_1 f_2$ is $O(g_1 g_2)$

Sorting Algorithms

ordered unordered

INSERTION SORT

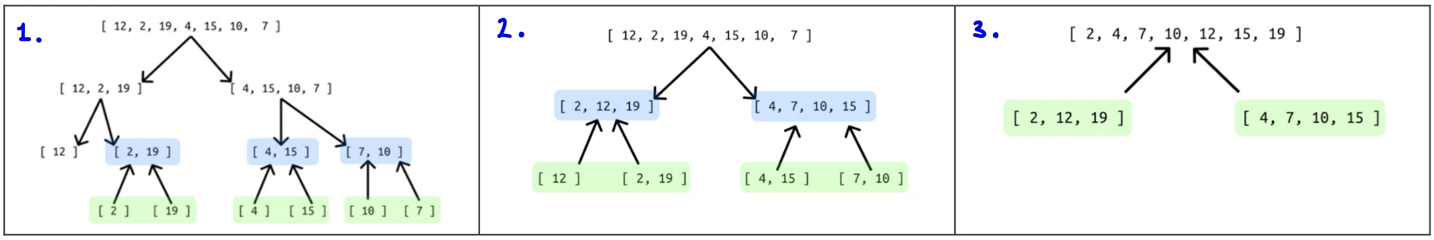
[71| 87, 37, 43, 39, 55] # 1st pass
[71, 87| 37, 43, 39, 55] # 2nd pass
[37, 71, 87| 43, 39, 55] # 3rd pass
[37, 43, 71, 87| 39, 55] # 4th pass
[37, 39, 43, 71, 87| 55] # 5th pass
[37, 39, 43, 55, 71, 87] # 6th pass

SELECTION SORT

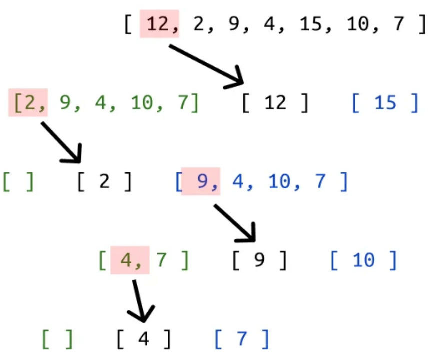
[71, 87, 37, 43, 39, 55]
[71, 55, 37, 43, 39, 87] # 1st pass
[39, 55, 37, 43, 71, 87] # 2nd pass
[39, 43, 37, 55, 71, 87] # 3rd pass
[39, 37, 43, 55, 71, 87] # 4th pass
[37, 39, 43, 55, 71, 87] # 5th pass

biggest value ordered part

MERGE SORT



QUICK SORT



------ CODE ------

Our pivot   Less than our pivot

Greater than our pivot

Sorting Algorithm Efficiency

	BEST	WORST	AVG
Insertion Sort	$O(n)$	$O(n^2)$	$O(n^2)$

→ Best: Array is already sorted
→ Worst: Array is sorted in reverse order

	BEST	WORST	AVG
Quick Sort	$O(n\log n)$	$O(n^2)$	$O(n\log n)$

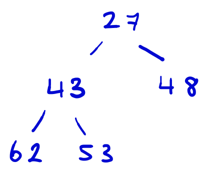
→ Best: Pivot divide equally
→ Worst: Array is already sorted

Selection Sort $O(n^2)$ | Merge Sort $O(n\log n)$

Heap Sort $O(n\log n)$

Prority Queues:

[0, 27, 43, 48, 62, 53] $\longrightarrow$



root_index = 1

left_child = 2 × parent_index

right_child = 2 × parent_index + 1

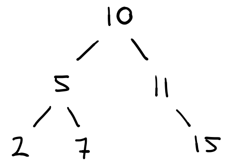
parent = child // 2

Tree Traversal

Pre-order: 10 5 2 7 11 15

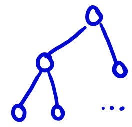
In-order: 2 5 7 10 11 15

Post-order: 2 7 5 15 11 10



Binary Heap

Left complete



All levels are full but the last, which fills from left to right

Solving Recurrences

Bottom-up Method

1) Start with base case
2) Compute first few values
3) Find Pattern

Example: $f(n) = n \times \left(\sqrt{f(n-1)} + 1\right)^2$ where $f(0) = 0$

n	0	1	2	3	4	5	6
f(n)	0	1	4	9	16	25	36

This looks like $f(n) = n^2$ $\longrightarrow$ now prove by induction

Top-down method

1) Start with definition
2) Expand a few times
3) Find Pattern

Example: $T(n) = 2T(n-1) + 1$ where $T(1) = 0$

$= 2(2T(n-2) + 1) + 1$

... repeat a few times

*3rd iteration $= 16\,T(n-4) + 8 + 4 + 2 + 1$

Hmm... Let $k = \#\,\text{iteration} + 1$, we can simplify the pattern as ...

$T(n) = 2^k\,T(n-k) + 2^k - 1$ # observation we reach base case when $n-k=1$

$\Rightarrow T(n) = 2^{n-1} - 1$ $\longrightarrow$ now prove with induction

Change of Variable Trick

1) Let $n = f(k)$
2) Let $U(k) = T(n) = T(f(k))$
3) Solve $U(k)$ instead

Example: $T(n) = 2T(n/2) + n$ where $T(1) = 0$

let $k = \log_2 n \Rightarrow n = 2^k$ | $U(k) = T(n) = T(2^k)$

$U(k) = 2T(n/2) + n = 2T(2^{k-1}) + 2^k = 2U(k-1) + 2^k$

where $U(0) = T(1) = 0$

Induction Proof

1) Start with proving base case — Does our new definition match the previous?
2) Inductive Hypothesis — Assume our new definition holds true for all possible inputs "k"
3) Inductive Step — Prove our statement holds true for "k+1"