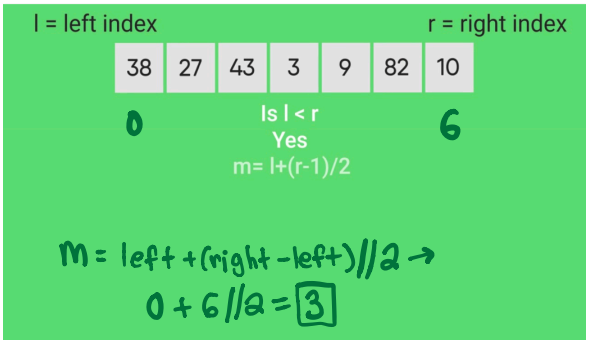
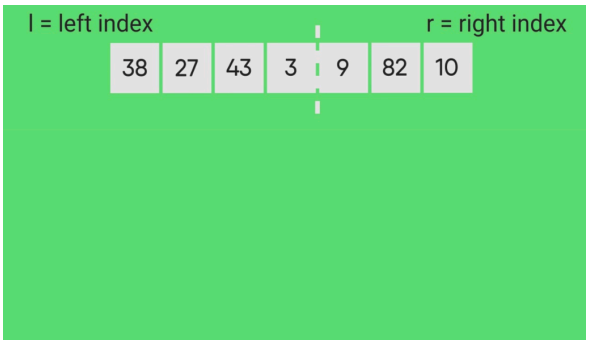
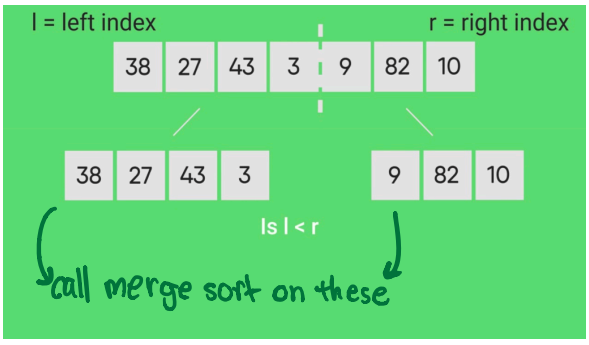
1

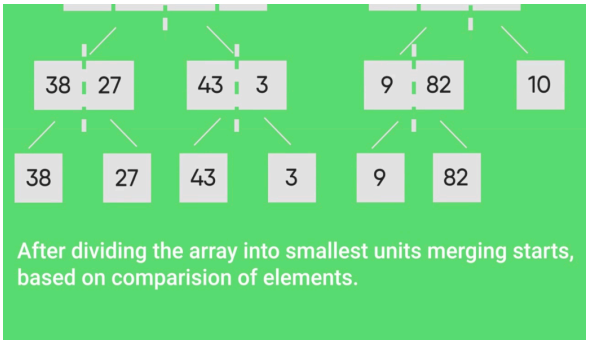
3-way merge sort: suppose that instead of dividing in half at each step of merge sort, you divide Into thirds, sort each third, and finally combine all of them using a three-way merge subroutine. What is the overall asymptotic running time of this algorithm? Note: merge step can still be implemented in O(n) time.

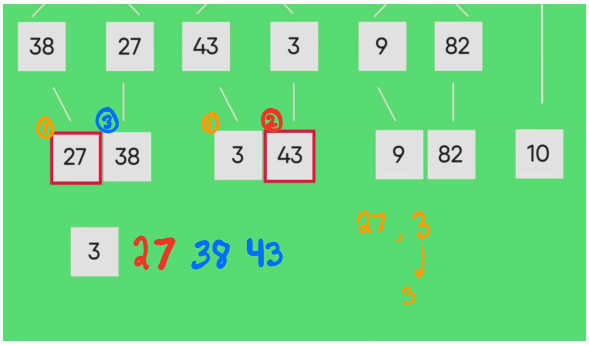
Merge Sort: O(n log n)

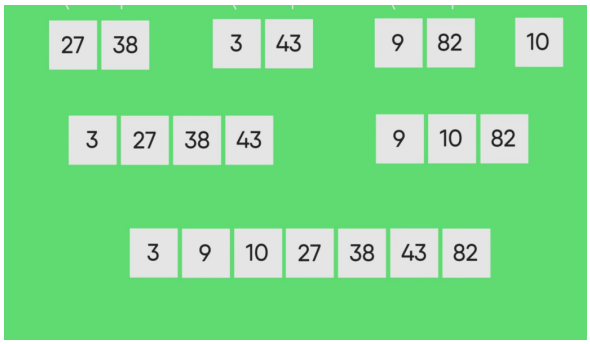












For this problem, height of recursion tree is $\log_3(n)$

* n for merge

nlogn

2

You are given functions f and g such that f(n) = O(g(n)). Is f(n) * log2(f(n)^c) = O(g(n) * log2(g(n)))? (here c is some positive constant). You should assume that f and g are nondecreasing and always bigger than 1.

$f(n) = O(g(n))$ $\quad f(n) * \log_2(f(n)^c) = O(g(n) * \log_2(g(n)))$

↳ translate to

$f(n) \leq c_1 g(n)$

↳ Let's derive a bound for the left side

$f(n) \leq c_1 \cdot g(n)$

$f(n)^c \leq (c_1 \cdot g(n))^c$

$\log_2(f(n)^c) \leq \log_2(c_1 \cdot g(n))^c$

$\log_2(f(n)^c) \leq c\log_2(c_1) + c\log_2(g(n))$

$f(n)$ $\qquad$ $c_1 g(n)$

$f(n)\log_2(f(n)^c) \leq \underbrace{c \cdot c_1 \cdot g(n)}_{d} \underbrace{\log_2(c_1)}_{} + \underbrace{c \cdot c_1}_{e} \cdot g(n) \cdot \log_2(g(n))$

$g(n) \cdot d \cdot \dfrac{\log_2(g(n))}{\log_2(g(n))} + e \cdot \log_2(g(n))$

$f(n) \cdot \log_2(f(n)^c) \leq \overbrace{\left(\dfrac{d}{\log_2(g(n))} + e\right)}^{c} g(n) \cdot \log_2(g(n))$

True

★ $\log(a \cdot b)^c =$

$c\log(a) + c\log(b)$

Method 2:

3

Assume again two (positive) nondecreasing functions f and g such that f(n) = O(g(n)). Is 2^f(n) = O(2^g(n))?

$f(n) = O(g(n))$

↳ $f(n) \leq c_1 \cdot g(n)$

$2^{f(n)} \leq c_1 \cdot 2^{g(n)}$

$2^{f(n)} - 2^{g(n)} \leq c_1$

$2^{(f(n) - g(n))} \leq c_1$

$2^{f(n)} = O(2^{g(n)})$

Simple checks:

$f(n) = g(n) = n$

$2^n \stackrel{?}{=} O(2^n)$ ✓

$f(n) = 10n \quad g(n) = n$

$2^{10n} \neq O(2^n)$ ✗

$2^{10n} \leq c 2^n$

$\rightarrow 2^{9n} \leq c \ldots$

True only when $f(n) - g(n) \leq 0$ or $f(n) \leq g(n)$

and: sometimes yes, sometimes no (depending on f and g)

4

K-way merge sort. Suppose you are given k sorted arrays, each with n elements, and you want to combine them into a single array of kn elements. Consider the following approach. Using the merge subroutine taught in lecture, you merge the first 2 arrays..then merge the 3rd given array with this merged version of the first two arrays, then merge the 4th given array with the merged version of the first three arrays, and so on until you merge in the final (kth) input array. What is the running time taken by this successive merging algorithm, as a function of k and n? Can you think of a faster way to do the k-way merge procedure?

$_k$K sorted arrays O(n) merges n elements

2 merge(K_1 K_2), n

3 merge((K_1, K_2), K_3), 2n

4 merge (((K_1, K_2), K_3), K_4) ..., 3n

n · $n + 2n + 3n + 4n \ldots + (k-1)n$

$n(1+2+3+4+\ldots k-1)$ ★ Summation rule! $1+2+3+\ldots+k = k(k+1)/2$

$n\left[(k-1)(k-1+1)/2\right]$

$n(k^2-k)/2 \rightarrow O\left(\frac{nk^2}{2} - \frac{nk}{2}\right)$ (main term)

$\boxed{O(nk^2)}$

improvement: input of k sorted arrays is the same as the last level of merge sort.

merge pairwise and get $O(n \log n)$

↳ or here nk elements so $O(nk \log nk)$

Arrange the following functions in increasing order of growth rate (with g(n) following f(n) in your list if and only if f(n) = O(g(n))

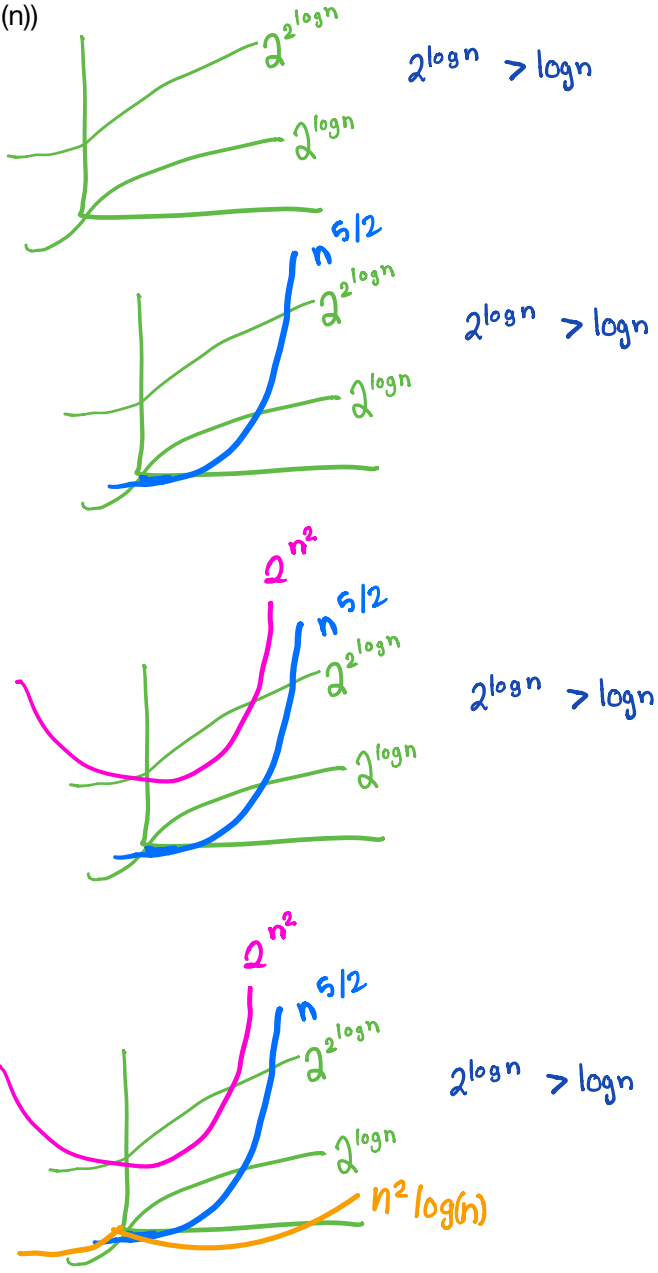
a) $2^{\log(n)}$

b) $2^{2\log(n)}$

c) $n^{5/2}$

d) 2^{n^2}

e) $n^2\log(n)$



Smallest to largest

aecbd ✓

Other:

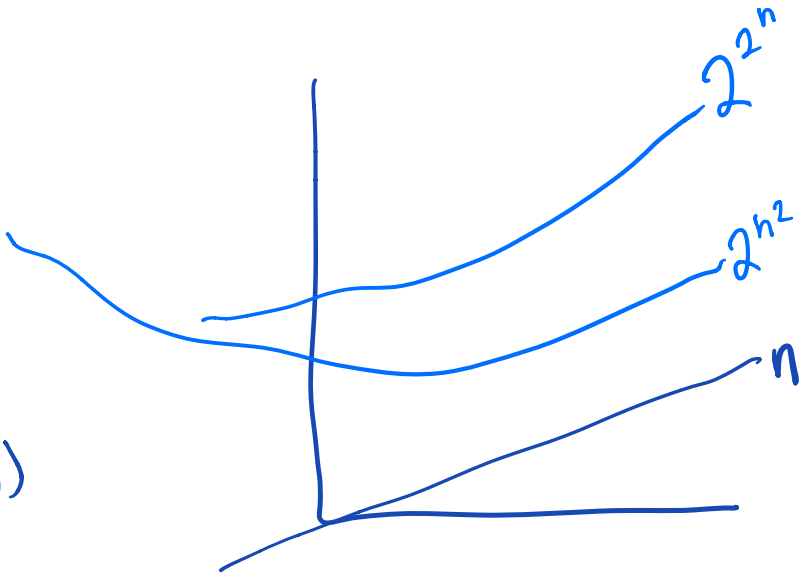
a) 2^{2^n}

b) 2^{n^2}

c) $n^2 \log(n)$

d) n

e) n^{2^n}



$n^{2^n} > 2^{2^n}$

Smallest to largest

n, $n^2 \log(n)$, 2^{n^2}, 2^{2^n}, n^{2^n}

d c b a e

✓