DSA

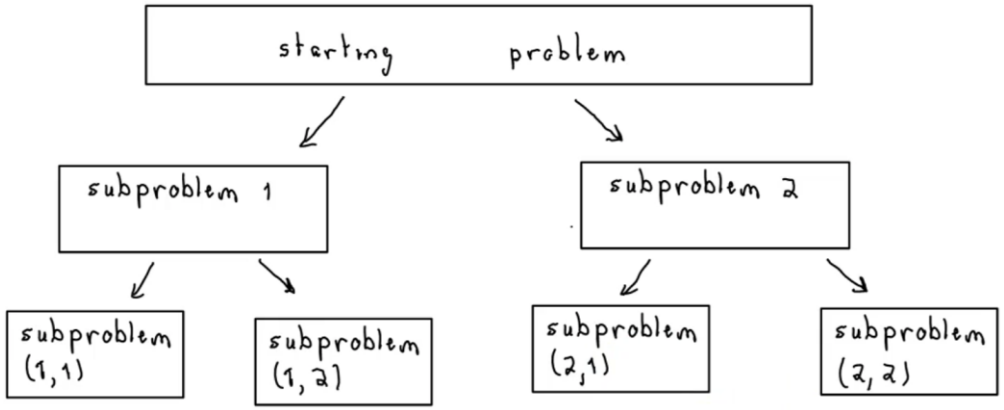
WEEK 3, DISCUSSIONS

Divide and conquer

Divide and conquer part seems quite intuitive, it was good to have actual examples of principle, I hope we look more into it during the class

Although this is already provided in one of the lectures, I'd also like to see a brief instruction-level example of another type of algorithm (not sorting) that utilizes the divide and conquer principle

https://www.freecodecamp.org/news/divide-and-conquer-algorithms/



Recursion:

- Smaller version of original problem
- Stop dividing when subproject has simple solution (base case)

O, Omega and Theta Notations

Distinguishing between O-notation, Omega-notation, and Theta-notation required careful attention, especially understanding which notation applies to the best, worst, or average case

I think it would be nice if some process for analyzing pseudocode was more clearly outlined.

Some aspects that still feel unclear are how to analyze the Pseudocode. I'd like to learn more about how to do that analysis step-by-step.

I still have a lot of trouble understanding the O-notation

When asking about efficiency, asymptotic efficiency or order of magnitude do they all mean finding the O of the algorithm when the data increased ?

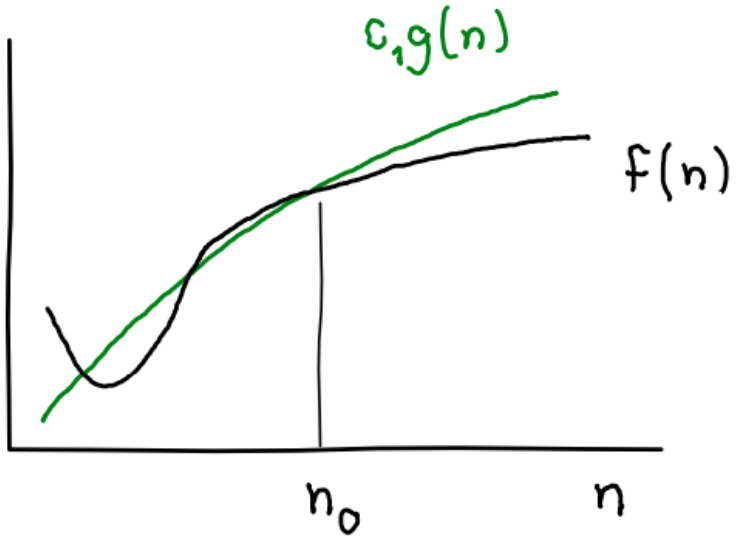
I didn't fully understand why one would want to calculate big O for the best case and big Omega for the worst case.

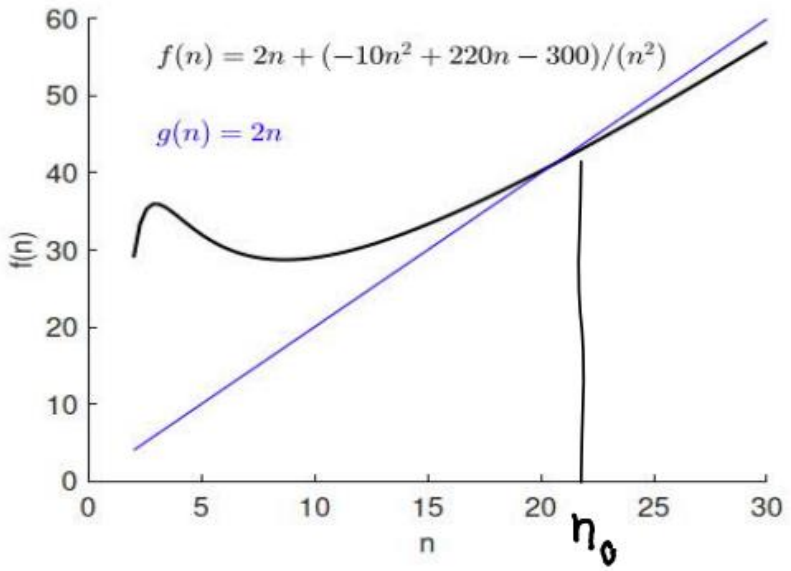
Understand what big-O and omega and theta means, but it remain unclear, how to define them for worst case and best case.

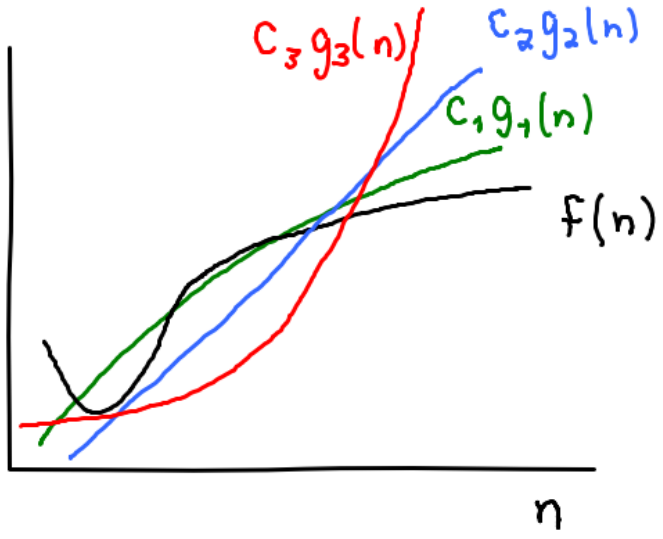
BUT, when going through videos about worst and best case, both of them includes Big-O and Big-Omega?

Asympthotical analysis: BigO

Find the fastest growing term of f(n), which is the closest

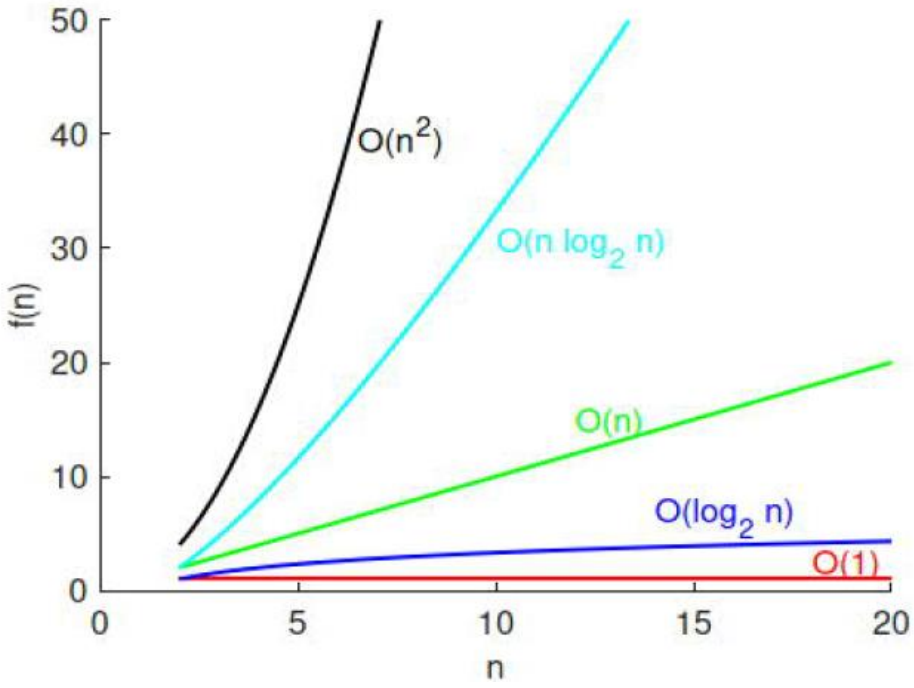






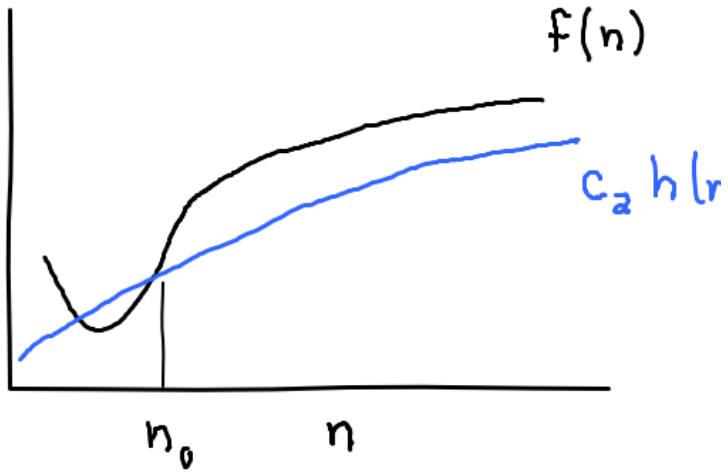
BigO - classes

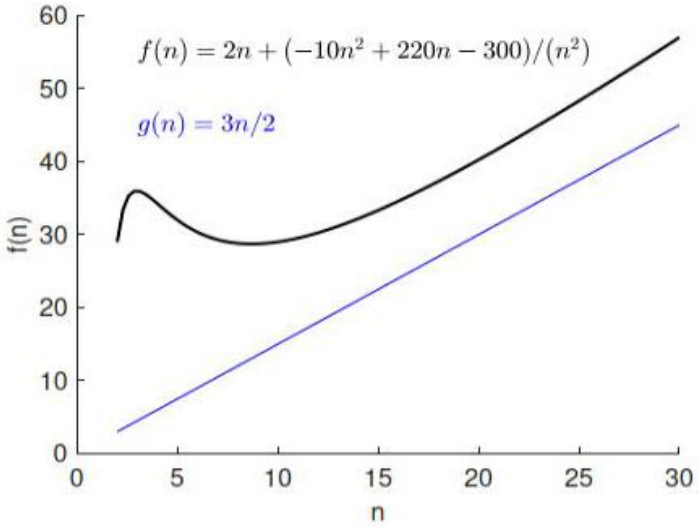
$O(g(n))$-class	name	if algorithm X's $f(n)$ belongs to $O(g(n))$
$O(1)$	constant time	Algorithm X runs in constant time.
$O(\log_2 n)$	logarithmic time	Algorithm X runs in log time.
$O(n)$	linear time	Algorithm X runs in linear time.
$O(n \log_2 n)$	linearithmic time	Algorithm X runs in linearithmic time.
$O(n^2)$	quadratic time	Algorithm X runs in quadratic time.

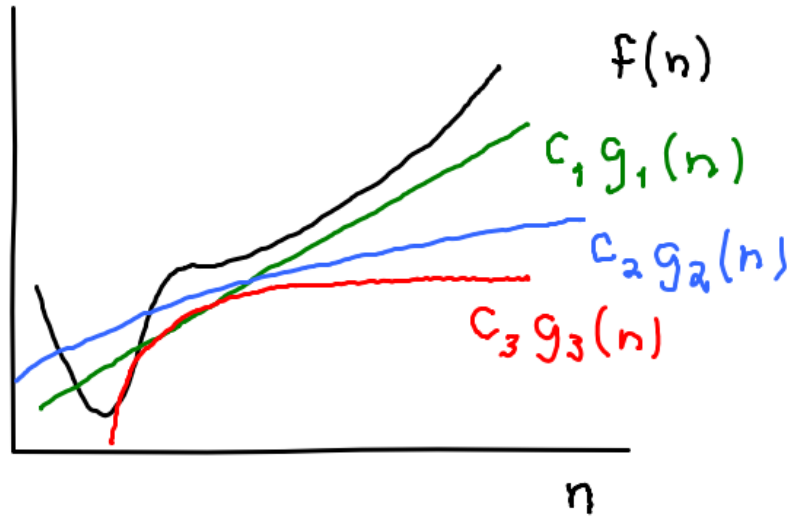


https://alyssaq.github.io/stl-complexities/

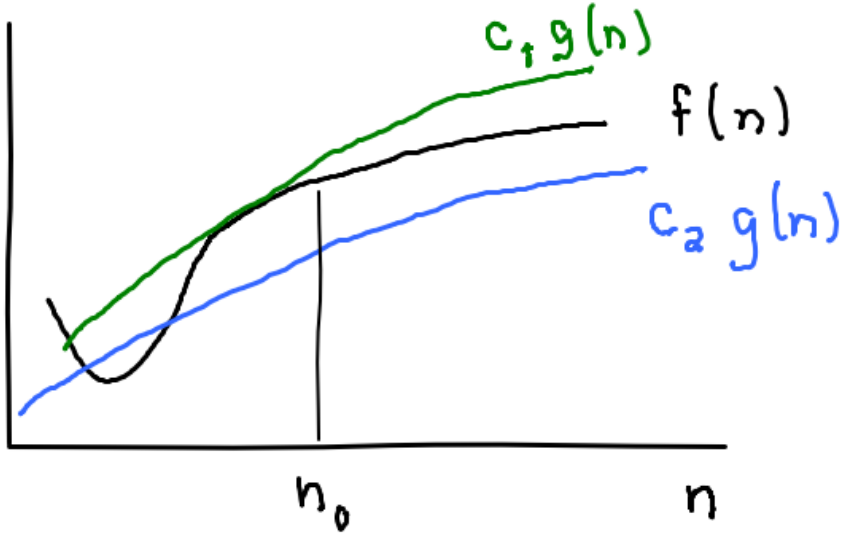
Asympthotical analysis: BigOmega, Ω(f(n))







Asympthotical analysis: BigTheta, Θ(f(n))



Partition + QuickSort

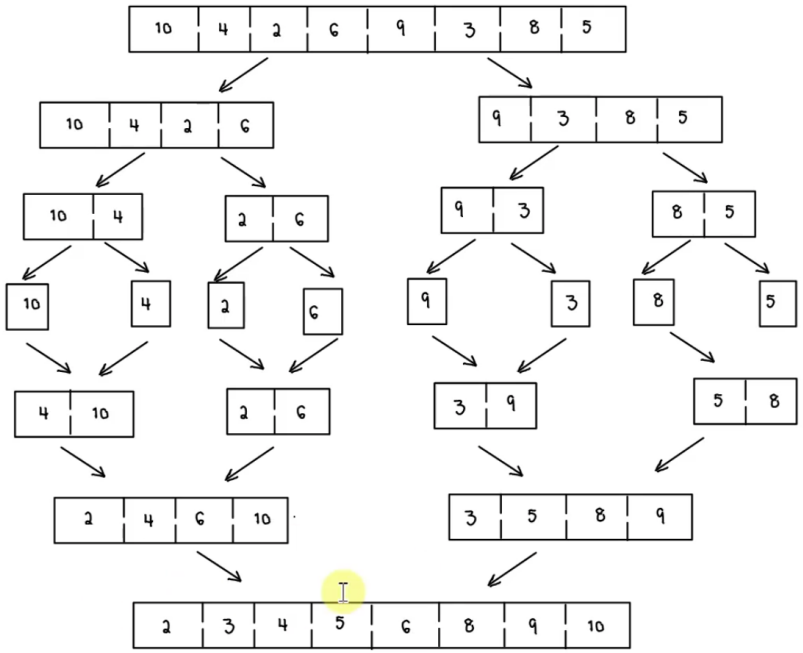
```
PARTITION(A, L, R)
   pivot=A[R]
   cut=L-1
   for j=L to R-1
      if A[j] <= pivot then
        cut=cut+1
        swap(A[cut], A[j])
      end
   end
   k=cut+1
   swap(A[k], A[R])
   return k
end
```

```
QUICKSORT(A, L, R)
   if L < R then
      k=PARTITION(A, L, R)
      QUICKSORT(A, L, k-1)
      QUICKSORT(A, k+1, R)
   end
end
```

Merge + MergeSort

```
MERGE(A, L, M, R)
   for i from L to R
         Temp[i]=A[i]
   end
   iL=L, iR=M+1, iA=L
   while iL <= M and iR <= R
      if Temp[iL] <= Temp[iR] then
        A[iA]=Temp[iL], iL=iL+1
      else
         A[iA]=Temp[iR], iR=iR+1
      end
      iA=iA+1
   end
   if iL > M then
      copy Temp[iR..R] to A[iA..R]
   else
      copy Temp[iL..M] to A[iA..R]
   end
 end
```

```
MERGESORT(A, L, R)
   if L < R then
      M=floor((L+R)/2)
      MERGESORT(A, L, M)
      MERGESORT(A, M+1, R)
      MERGE(A, L, M, R)
   end
end
```



Comparing sorting algorithms

How the performance of Quicksort compares to Mergesort in all practical scenarios, especially considering factors like memory usage and the state of the input data

Understanding the analyses of different sorting algorithms seems a bit challenging, particularly in understanding the assumptions at first

Also, I’m curious why are Quicksort(when considering in the best case) and Mergesort theoretically equally efficient, but Quicksort usually performs better in practice?

Please elaborate on quicksort and asymptotic analysis.

Up to this point I haven't fully understood why mergesort would work better than other algorithms in high n, so I guess I'll need to go further into that

I would love to understand and compare the built-in sort() function of C++ with these.

Best, worst and average cases

I am still confused as to how to determine exactly how to determine what the time complexity (best / worst case) for an algorithm.

Specifically, tracing the recursion trees and summing the operations at each level to prove that both algorithms are Θ(n log n) in their best/average cases required careful thought. The assumption of having an input size of a power of two (n=2^x) initially seemed like a simplification, but understanding why the conclusion holds for any n was a key insight.

Aspects that remain a bit unclear are the precise probabilistic analysis for Quicksort's average case and the different methods for choosing a pivot to avoid the $O(n^2)$ worst-case scenario.

In Algorithm efficiency (the best, the worst and the average case), and analysis of quicksort and mergesort, n is different 2^x -1, which I did not understand where it came from, therefore those 3 videos were difficult to follow. So, I would like to learn about this topic more.

I'd appreciate more analyses and a framework of how to do analysis to be provided. In video 05-06, around 7:30, I don't understand why the runtime efficiency of lines 8-21 of algorithm MERGE is 2^x.

One topic that is still a bit unclear is the average-case analysis. The lectures mentioned that in practice, Quicksort is often better than Mergesort, even though their worst-case (or best-case for Quicksort) efficiencies look similar. I am very curious to learn more about why this is the case and how we can analyze the "average" performance of an algorithm.

Best, worst and average case

What is the input that gives the worst case behaviour?

Average case analysis is difficult, since we need to make assumptions about the probability distribution of the input data.

Binarysearch

- Best case where you do do while loop just once, constant time, $O(1)$, $\Omega(1)$, $\Theta(1)$
- Worst case $O(\log_2 n)$, $\Omega(\log_2 n)$, $\Theta(\log_2 n)$ (how many cases we need to divide n to 2, until there is just one item left)

$$\frac{n}{2^k} = 1 \Rightarrow 2^k = n \Rightarrow k = \log_2 n$$

Best, worst and average case

InsertionSort

- Best case, where you do newer do while loop, linear, O(n), $\Omega(n)$, $\Theta(n)$ (e.g. when the input array is already in order)
- Worst case, when the while loop is done the max times, $O(n^2)$, $\Omega(n^2)$, $\Theta(n^2)$ (e.g. when input array is in reverse order)

MergeSort

- We select A[1..2^x], merge will do 2^x operations (for loop & while loop per each element), e.g. $\Theta(2^x)$
- MergeSort has *x* recursion level, and on each merge does 2^x operations, e.g. x * 2^x , when n= 2^x we can write the function as: x = $log_2 n$, so $log_2 n * n = nlog_2 n$
- $\Theta(nlog_2 n)$

Best, worst and average case

QuickSort

- We select A[1..n], where n= 2^x -1, then partition will do 2^x -2 operations (for loop for each element), e.g. $\Theta(2^x)$
- QuickSort has *x* recursion level (assuming pivot is always at the middle), and on each partition does 2^x operations, e.g. x * 2^x , when n= 2^x we can write the function as: x = $log_2 n$, so $log_2 n * n = nlog_2 n$
- Best case: $\Theta(nlog_2 n)$

Comparison of sorting algorithms

	Worst *)	Average/expected *)	Best
InsertionSort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n)$
QuickSort	$\Theta(n^2)$	$\Theta(nlog_2n)$	$\Theta(nlog_2n)$
MergeSort	$\Theta(nlog_2n)$	$\Theta(nlog_2n)$	$\Theta(nlog_2n)$

*) Introduction to Algorithms, Cormen, Leiserson, Rivest, Stein

Clarification

When stating asympthotic running time estimate it needs to be considered:

- are we talking about
  - **worst case (important for safety purposes)**
  - **average case (the most relevant in practice)**
  - best case (important in analysing encryption)

- another thing is how well we can prove mathematically, how the function behaves e.g. how precise is our knowledge:
  - BigO is the upperbound (it is not worse than this)
  - Omega (Ω) is the lowerbound (it is newer better than this)
  - Theta (Θ) means, this is the right function, just the coefficient changes

You need to understand the context

INSERTION SORT:

- the worst case of insertion sort, we have $O(n^2)$, (reverse order)
  - with math we can figure out a case that the worst case is also $\Omega(n^2)$, so the worst case is $\Theta(n^2)$

- the best case of insertion sort, we can say that it is O(n), (already in order)
  - with math we can figure out that it is also also $\Omega(n)$, so the best case is $\Theta(n)$

- we can say that in ALL cases it is $O(n^2)$ and $\Omega(n)$ but in ALL cases there is no Θ

- the average case is the same as worst case (in this case, not always)

NOTE 1: an algorithm can be both $O(n^2)$ and $\Theta(n)$ at the same time, since BigO is just the upperbound, when comparing algorithms you should use Θ, if it exist

Exercise: Mystery sort

```
/* a[0] to a[aLength-1] is the array to sort */
int i,j;
int aLength; // initialise to a's length

/* advance the position through the entire array */
/*   (could do i < aLength-1 because single element is also min element) */
for (i = 0; i < aLength-1; i++)
{
    /* find the min element in the unsorted a[i .. aLength-1] */

    /* assume the min is the first element */
    int jMin = i;
    /* test against elements after i to find the smallest */
    for (j = i+1; j < aLength; j++)
    {
        /* if this element is less, then it is the new minimum */
        if (a[j] < a[jMin])
        {
            /* found new minimum; remember its index */
            jMin = j;
        }
    }

    if (jMin != i)
    {
        swap(&a[i], &a[jMin]);
    }
}
```

https://en.wikipedia.org/wiki/Selection_sort

Exercises

I want to ask about question 1 and 3 of Asymptotic analysis and hope we would be able to discuss the answers to it next session

This question "After comparing the pseudocodes of the sorting algorithms, you would be able to say that..." obviously has very confusing options.

- For example, the answer "Insertionsort is the most inefficient sorting algorithm." refers to all existing algorithms or only among those that were considered on week 2?
- Then the option "Mergesort is always as fast as always in spite of the data." means that in spite the volume of data or the content/order of the data?
- Then the option "N log N is the most common order of magnitude for sorting algorithms." refers to those algorithms considered on this week only (Mergesort and Quicksort) or all algorithms?