CS32 Intro to Computer Science II

Baoxiong Jia & Muthu Palaniappan, DIS 1C Week 7
UCLA Spring 2021

About Us

- TA: Baoxiong Jia
  - Email: baoxiongjia@cs.ucla.edu
  - Office Hours: Tuesday 8:30-10:30am
  - Thursday 8:30-10:30am
  - Discussion 1C: Friday 12:00-13:50pm

- LA: Muthu Palaniappan
  - Email: muthupal@g.ucla.edu
  - Office Hours: Monday 10:30-11:30am
  - Wednesday 10:30-11:30am

Outline

- Algorithm complexity
- Big-O notation
- Sorting algorithms

Big-O notation

- Describes how fast an algorithm is (asymptotic time complexity)
  - Care more about how many basic operations will be executed when the scope (value range) of the problem increases
  - Mainly caused by loops and recursion for time complexity
    - addition for parallel loops
    - multiplication for nested loops

```
void bar( int n, int q )
{
   for (int i=0 ; i < n*n ; i++)
   {
       for (int j = 0; j < q; j++)
            cout << "I love CS!";
   }
}

void bletch( int n, int q )
{
   for (int i=0 ; i < n ; i++)
        cout << "Muahahaha!";

   for (int i=0 ; i < q*q ; i++)
        cout << "Vomit!";
}
```

```
void burp( int n )
{
   for (int i=0 ; i < n ; i++)
        cout << "Muahahaha!";

   for (int i=0 ; i < n*n ; i++)
        cout << "Vomit!";
}

void barf( int n, int q )
{
   for (int i=0 ; i < n ; i++)
   {
       if (i == n/2)
       {
            for ( int k = 0; k < q; k++ )
                cout << "Muahahaha!";
       }
       else
            cout << "Burp!";
   }
}
```

Big-O notation

- For recursive functions, we need to consider the number of sub-problems we divide into and consider basic operations inside each base case.

  Time Complexity = Number of recursive functions * cost per recursive call

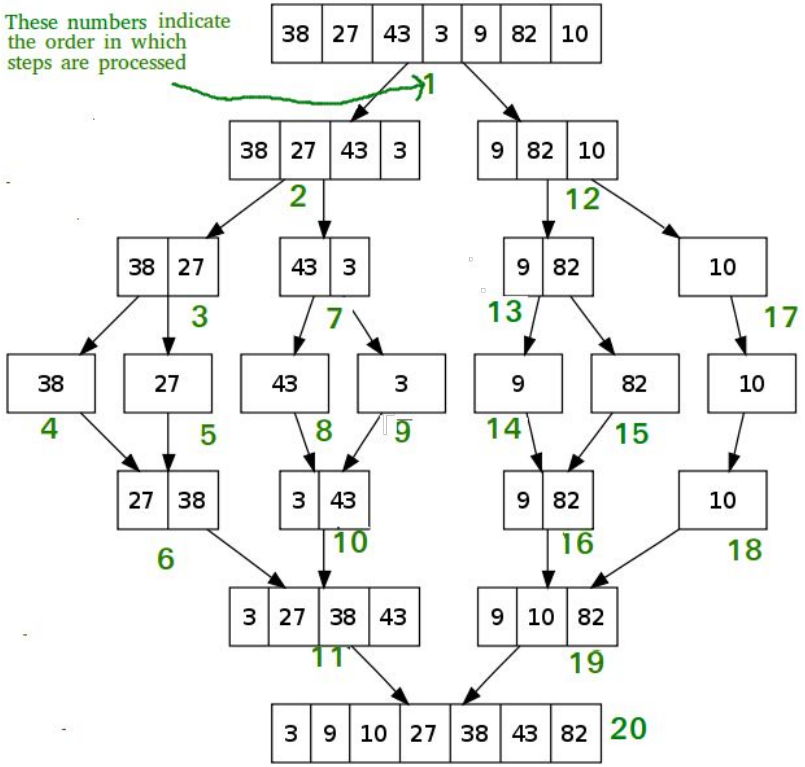
Merge-Sort

```
void mergeSort(int arr[], int l, int r)
{
    if (l < r)
    {
        // Same as (l+r)/2, but avoids overflow for
        // large l and h
        int m = l+(r-l)/2;

        // Sort first and second halves
        mergeSort(arr, l, m);
        mergeSort(arr, m+1, r);

        merge(arr, l, m, r);
    }
}
```

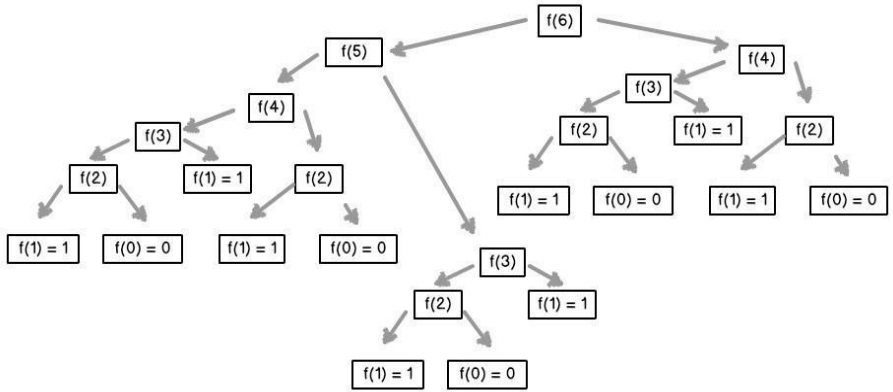
O(nlogn) https://repl.it/@jiajerry/MergeSort



For recursive functions

Fibonacci(n) = Fibonacci(n - 1) + Fibonacci(n - 2)

https://repl.it/@jiajerry/Fibonacci



T(n) = T(n - 1) + T(n - 2) + C(time for addition)

= 2T(n - 2) + C

= 2*(2*T(n - 4) + C) + C = 4T(n - 4) + 3C

= 8T(n - 6) + 7C

= 2^k * T(n – 2k) + (2^k - 1)C

Base case T(0) = T(1) = C(returning value)

T(n) = $2^{\frac{n}{2}}$ * T(0) + ($2^{\frac{n}{2}}$ - 1)C = $2^{\frac{n}{2}}$ (1 + C) – C => O($2^{\frac{n}{2}}$)

Sorting algorithms

Sort Name	Stable/ Non-stable	Notes
Selection Sort	Unstable	Always $O(n^2)$, but simple to implement. Can be used with linked lists. Minimizes the number of item-swaps (important if swaps are slow)
Insertion Sort	Stable	$O(n)$ for already or nearly-ordered arrays. $O(n^2)$ otherwise. Can be used with linked lists. Easy to implement.
Bubble Sort	Stable	$O(n)$ for already or nearly-ordered arrays (with a good implementation). $O(n^2)$ otherwise. Can be used with linked lists. Easy to implement. Rarely a good answer on an interview!
Shell Sort	Unstable	$O(n^{1.25})$ approx. OK for linked lists. Used in some embedded systems (eg, in a car) instead of quicksort due to fixed RAM usage.
Quick Sort	Unstable	$O(n \log_2 n)$ average, $O(n^2)$ for already/mostly/reverse ordered arrays or arrays with the same value repeated many times. Can be used with linked lists. Can be parallelized across multiple cores. Can require up $O(n)$ slots of extra RAM (for recursion) in the worst case, $O(\log_2 n)$ avg.
Merge Sort	Stable	$O(n \log_2 n)$ always. Used for sorting large amounts of data on disk (aka "external sorting"). Can be used to sort linked lists. Can be parallelized across multiple cores. Downside: Requires n slots of extra memory/disk for merging – other sorts don't need xtra RAM.
Heap Sort	Unstable	$O(n \log_2 n)$ always. Sometimes used in low-RAM embedded systems because of its performance/low memory req'ts.

https://www.cs.usfca.edu/~galles/visualization/ComparisonSort.html
https://www.toptal.com/developers/sorting-algorithms