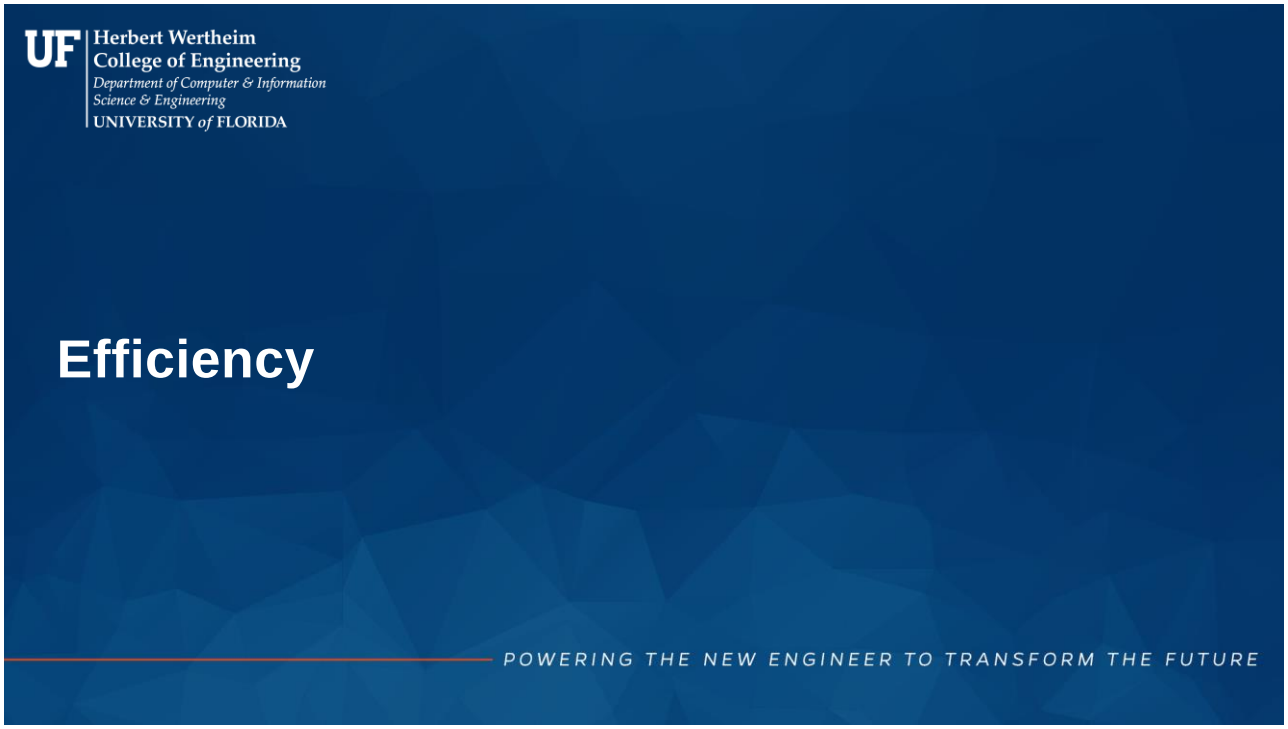


To Do list

- Questionnaire
  - On Canvas: https://ufl.instructure.com/courses/487242/quizzes/1281313
- Typing speed
  - typing test at http://www.typingtest.com
- Vjudge account
  - Create your account using your **full name** and join the group:
  - https://vjudge.net/group/cis4930fal23
- Discord group
  - https://discord.gg/Y6WHuybA

What is an algorithm

- A set of steps to accomplish a task
- Example
  - GPS
    - Shortest path
  - Online shopping payment
    - Encryption
  - Package delivery
    - Complicated routing
- Precise description to solve computational problems
  - Correctness
    - Approximation
  - Efficiency
    - Running time

Time complexity

- Determining the number of steps (operations) needed as a function of the problem size
- Asymptotic Analysis
  - O(f(n)), Big-Oh of f of n, the Asymptotic Upper Bound
  - Ω(f(n)), Big-Omega of f of n, the Asymptotic Lower Bound
  - Θ(f(n)), Big-Theta of f of n, the Asymptotic Tight Bound
- Which one may be faster?

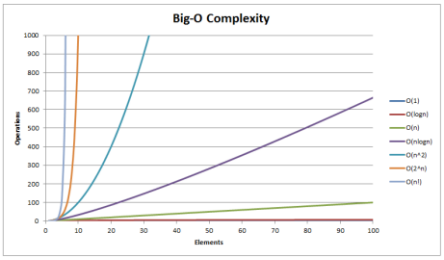
```
fact = 1;
for(i = 1; i <= n; i++)
  fact *= i;
```

(a)

```
sum = 0;
for(i = 1; i <= n; i++)
  for(j = 1; j <= n; j++)
    sum += a[i][j];
```

(b)

Example



Algorithm	Best Time Complexity	Average Time Complexity	Worst Time Complexity	Worst Space Complexity
Linear Search	O(1)	O(n)	O(n)	O(1)
Binary Search	O(1)	O(log n)	O(log n)	O(1)
Bubble Sort	O(n)	O(n^2)	O(n^2)	O(1)
Selection Sort	O(n^2)	O(n^2)	O(n^2)	O(1)
Insertion Sort	O(n)	O(n^2)	O(n^2)	O(1)
Merge Sort	O(nlogn)	O(nlogn)	O(nlogn)	O(n)
Quick Sort	O(nlogn)	O(nlogn)	O(n^2)	O(log n)
Heap Sort	O(nlogn)	O(nlogn)	O(nlogn)	O(n)
Bucket Sort	O(n+k)	O(n+k)	O(n^2)	O(n)

- Question: which one is better, $O(n*n!)$ or $O(n^2*2^n)$?
  - C++ STL **algorithm:: sort**
  - Java **Collections.sort**
  - Python **list.sort() and sorted()**
  - Object-oriented approach

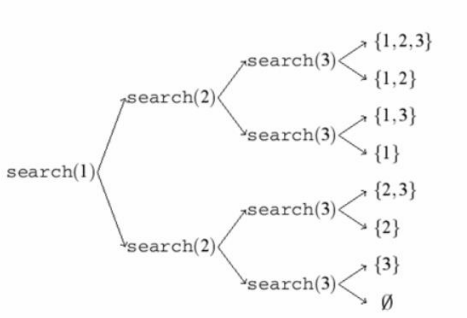
https://www.hackerearth.com/practice/notes/sorting-and-searching-algorithms-time-complexities-cheat-sheet/

Estimating Efficiency

Input size	Expected time complexity
$n \leq 10$	$O(n!)$
$n \leq 20$	$O(2^n)$
$n \leq 500$	$O(n^3)$
$n \leq 5000$	$O(n^2)$
$n \leq 10^6$	$O(n \log n)$ or $O(n)$
n is large	$O(1)$ or $O(\log n)$

What is the time complexity?

```
void search(int k) {
      if (k == n+1) {
            // process subset
      } else {
            // include k in the subset
            subset.push_back(k);
            search(k+1);
            subset.pop_back();
            // don't include k in the subset
            search(k+1);
      }
}
```



Time complexity for string operations

Operation	Time complexity
appending additional characters at the end of its current value	generally up to linear in new String length
Remove part of the string, reducing its length	generally up to linear in new String length
Returns a reference to the character at position	Constant
accesses the last character	Constant
Inserts additional characters into the string	generally up to linear in new String length
+	generally linear
Get a substring	generally linear
Find a character in the string	generally linear

Example: UVA10684 - The jackpot

- Question

  As Manuel wants to get rich fast and without too much work, he decided to make a career in gambling. Initially, he plans to study the gains and losses of players, so that, he can identify patterns of consecutive wins and elaborate a win-win strategy. But Manuel, as smart as he thinks he is, does not know how to program computers. So he hired you to write programs that will assist him in elaborating his strategy.

  Your first task is to write a program that identifies the maximum possible gain out of a sequence of bets. A bet is an amount of money and is either winning (and this is recorded as a positive value), or losing (and this is recorded as a negative value).

- Input

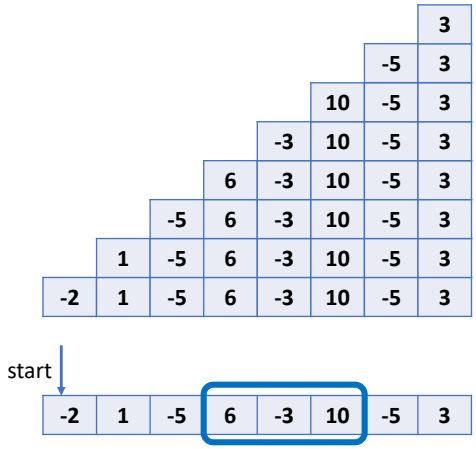
  The input set consists of a positive number N ≤ 10000 , that gives the length of the sequence, followed by N integers. Each bet is an integer greater than 0 and less than 1000. The input is terminated with N = 0.

- Output

  For each given input set, the output will echo a line with the corresponding solution. If the sequence shows no possibility to win money, then the output is the message 'Losing streak.'

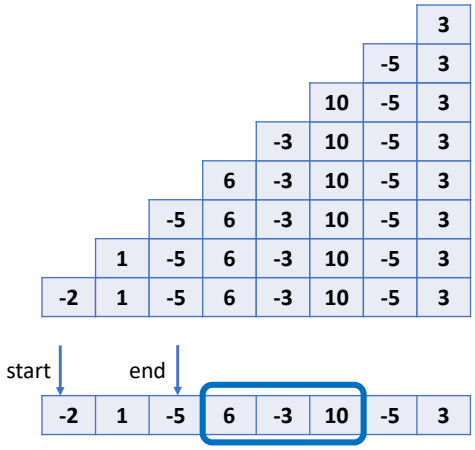
Largest Sum Continuous Subarray

- Problem
  - Given an array **a** of **N** integers. Find the continuous sub-array with maximum sum.
  - Constraints:
    $1 \le N \le 10^4$
    $-10^3 \le a[i] <= 10^3$
- Solutions
  - Naïve $O(n^3)$



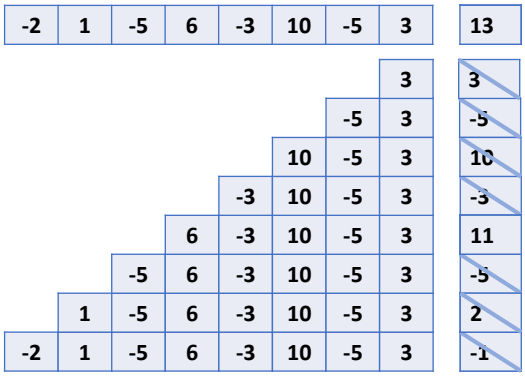
Largest Sum Continuous Subarray

- Problem
  - Given an array **a** of **N** integers. Find the continuous sub-array with maximum sum.
  - Constraints:
    $1 \le N \le 10^4$
    $-10^7 \le a[i] <= 10^7$
- Solutions
  - Naïve $O(n^3)$
  - Easy adjustment $O(n^2)$
  - Best: Kadane's Algorithm O(n)
  - Range of integer



Kadane's Algorithm

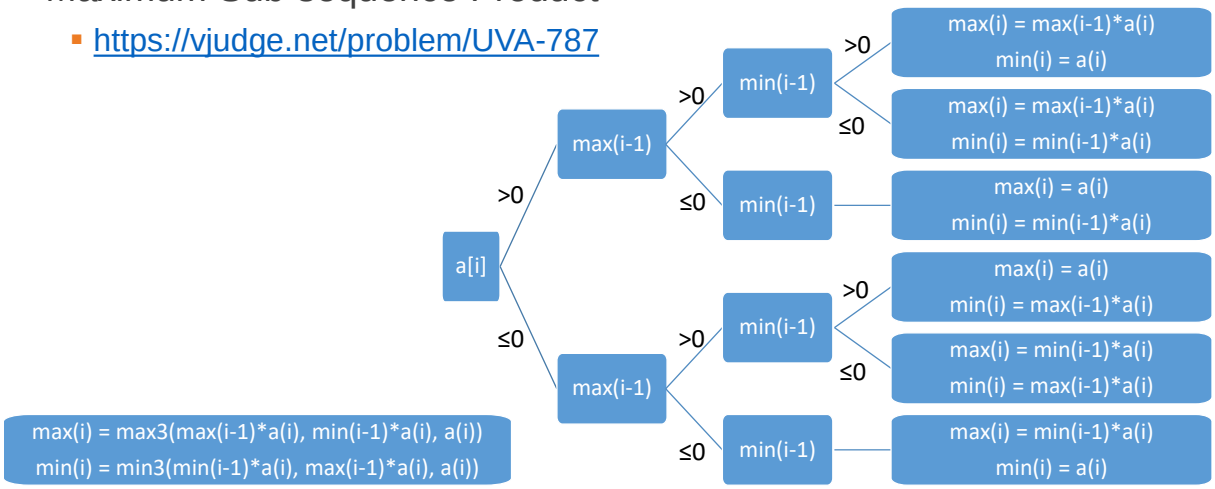
- Observation:
  - If you know a maximum subarray of a[1 : : j], extend the answer to find a maximum subarray ending at index j+1:
    - a maximum subarray of a[1 : : j]
    - or a subarray a[i : : j +1], for some $1 \le i \le j + 1$
      - The sum of the new subarray was the maximum subarray of a[1 : : j] + a[j+1]




```
int maxSum = 0, currentSum = 0;
for (int i = 0; i < n; i++) {
   currentSum = max(a[i], currentSum + a [i]);
   maxSum = max(currentSum, maxSum);
}
```

Extension

- Maximum Sub-sequence Product
  - https://vjudge.net/problem/UVA-787



Catch: big numbers

Code optimization

- Implementation is also important

```
bool ok = false;
for (int i = 0; i < n; i++) {
    if (a[i] == x) ok = true;
}
```

- More efficient

```
bool ok = false;
for (int i = 0; i < n; i++) {
    if (a[i] == x) {ok = true; break;}
}
```

- Not all optimizations are useful

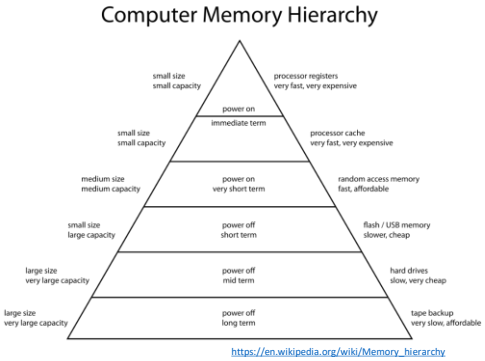
```
a[n] = x;
int i;
bool ok = false;
for (i = 0; a[i] != x; i++);
if (i < n) ok = true;
```

Processor features

- Asymptotic analysis does not account for differences in memory access times
  - Programs that do more work may take less time than those that do less work
  - Cache friendly

```
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
        s += x[i][j];
    }
}
```

```
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
        s += x[j][i];
    }
}
```



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

- Parallelism

```
ll f = 1;
for (int i = 1; i <= n; i++) {
    f = (f*i)%M;
}
```

```
ll f1 = 1;
ll f2 = 1;
for (int i = 1; i <= n; i += 2) {
    f1 = (f1*i)%M;
    f2 = (f2*(i+1))%M;
}
ll f = f1*f2%M;
```