Advanced Computer Contest Preparation

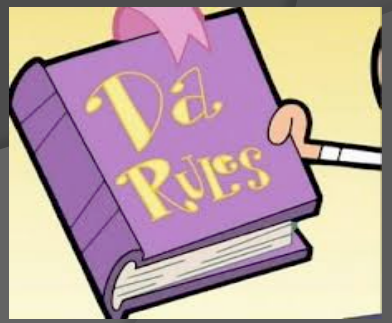
Lecture 2

CONTEST STRATEGIES AND ALGORITHM EFFICIENCY

Contest Strategies

Read Contest Instructions!

- Know the contest rules!
- Know whether you use standard input/output, or read/write to file
- How much time you have to finish the contest
- Know the difficulty and number of problems



Match Output!

- Match output **EXACTLY**
- Pay attention to output format given
- Remove all debugging output
  - Printing out values to see how they change
- Remove all unnecessary input prompts
  - "Please input an integer:"
- Remove all unnecessary output prompts
  - "Your answer is:"

```
 1  #include <iostream>
 2
 3  using namespace std;
 4
 5  int main()
 6  {
 7      int a,b;
 8      cin >> a >> b;
 9      cout << a+b;
10      return 0;
11  }
```

Answer Queries Immediately!

- Some problems contain “queries”
  - Query means “question”
- Example:
  - Q queries, print the sum of two integers per query
    - Example: 3 queries: 1 and 2, 3 and 10, 5 and 2
- Print answer to queries immediately after they are computed
- You DO NOT NEED to store answers in order to print them all at the end at the same time

```
#include <iostream>
#include <cstdio>

using namespace std;

int N,A,B;

int main()
{
     scanf("%d",&N);
     while(N--){
          scanf("%d%d",&A,&B);
          printf("%d\n",A+B);
     }
     return 0;
}
```

Start From Beginning!

- Ties **might** be broken by time
- Guarantee that you will get some points, rather than none
- Better to spend 2 hours on last problem while finishing the others than to spend all 3 hours on last problem

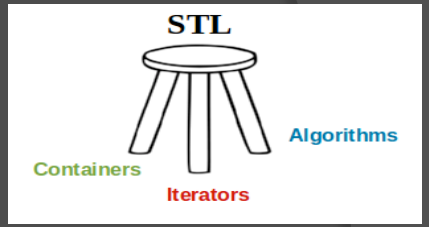
1 100	2 100	3 100	4 100	5 100	Points
100 00:50:10	100 01:20:33	100 01:36:34	100 03:32:50	100 03:19:43	500
100 00:04:29	100 00:22:02	100 00:42:58	0 02:35:38	80 02:50:18	380
100 00:01:26	100 00:33:07	100 00:40:50	30 03:59:19	10 03:46:06	340
100 00:25:19	100 01:02:13	30 02:41:19	70 02:27:50	10 03:11:51	310
100 00:07:55	100 00:37:59	100 00:45:40		0 03:56:29	300
100 00:06:01	100 00:59:28	100 01:53:10	0 03:57:39		300
100 00:58:20	100 02:37:17	100 01:33:39	0 03:56:30		300
100 01:02:32	100 03:56:30	100 01:43:44			300
100 02:10:25	100 03:16:35	10 03:42:46			210

Use C++!

- Not mandatory
- Contest problems are designed for C++
  - Poor C++ solution can be faster than correct Python solution
- C++'s speed allows you to get some points by brute forcing problems you don't know how to solve

Know Your Language!

- Know all features of your language
- Know how to use built-in libraries and functions
- Know how to use documentation/ references
- Know how to get standard input/output
- Know how to read/write files
- e.g. C++: Know how to use **`printf`**/ **`scanf`**
- e.g. Java: Know how to use **`BufferedReader`**



Prepare Before the Contest!

- If possible:
  - Create all files necessary for the contest BEFORE start
  - Type code required for every file (e.g. main function, imports)
  - Have references easily accessible

Use a Template!

- A **template** is a snippet of code that provides "shortcuts" for frequently used code
  - e.g. input/output, pairs, imports, etc.
- Use templates to save time when using these frequent operations
- Keep the template and copy/paste to your programs before starting
  - Make sure contest rules allow you to do so

Sample C++ Template

```
#include<bits/stdc++.h>
using namespace std;

#define s(n)                        scanf("%d",&n)
#define sc(n)                       scanf("%c",&n)
#define ss(n)                       scanf("%s",n)
#define INF                         (int)2e9
#define LLINF                       LLONG_MAX
#define x                           first
#define y                           second
#define pb                          push_back
#define fill(a,v)                   memset(a, v, sizeof a)
#define mp                          make_pair
#define all(a)                      a.begin(), a.end()
#define COMPRESS(a)                 sort(all(a)); a.erase(unique(all(a)),a.end())
#define IOS                         ios::sync_with_stdio(0); cin.tie(0)
#define debug(args...)              {vector<string> v = split(#args, ',');err(v.begin(), args);}
#define bits(a)                     __builtin_popcount(a)
#define ffs(a)                      __builtin_ffs(a)
#define maX(a,b)                     ( (a) > (b) ? (a) : (b))
#define miN(a,b)                     ( (a) < (b) ? (a) : (b))

typedef long long ll;
typedef pair<int,int> pii;
typedef pair<long long, long long> pll;

vector<string> split(const string& s, char c){vector<string> v;stringstream st(s);string x;while(getline(st, x, c))v.pb(x);return move(v);}
void err(vector<string>::iterator it) {}
template<typename T, typename... Args>
void err(vector<string>::iterator it, T a, Args... args) {cerr<<it->substr((*it)[0]==' ',it->length())<<" = "<<a<<'\n';err(++it, args...);}
template <typename T1, typename T2>
inline ostream&operator<<(ostream&os,const pair<T1, T2>&p){return os<<"("<<p.x<<", "<<p.y<<")";}
template<typename T>
inline ostream&operator<<(ostream&os,const vector<T>&v){bool _=1;os<<"[";for(auto i:v){if(!_)os<<", ";os<<i;_= 0;}return os<<"]";}
template<typename T>
inline ostream&operator<<(ostream&os,const set<T>&v){os<<"-BEGIN SET-\n";bool _=1;for(auto i:v){if(!_)os<<"\n";os<<i;_= 0;}return os<<"\n-END SET-";}
template<typename T>
inline ostream&operator<<(ostream&os,const unordered_set<T>&v){os<<"-BEGIN SET-\n";bool _= 1;for(auto i:v){if(!_)os<<"\n";os<<i;_= 0;}return os<<"\n--END SET--";}
template<typename T1, typename T2>
inline ostream&operator<<(ostream&os,const map<T1,T2>&v){os<<"-BEGIN MAP-\n";bool _= 1;for(auto i:v){if(!_)os << "\n";os<<" "<<i;_=0;}return os<<"\n"<<"--END MAP--";}
template<typename T1, typename T2>
inline ostream&operator<<(ostream&os,const unordered_map<T1,T2>&v){os<<"-BEGIN MAP-\n";bool _= 1;for(auto i:v){if(!_)os<<endl;os<<" "<<i;_=0;}return os<<"\n--END MAP--";}
```

Sample BufferedReader (Java) Template

```
static BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
static StringTokenizer st;

static String next() throws IOException {
    while (st == null || !st.hasMoreTokens())
        st = new StringTokenizer(br.readLine().trim());
    return st.nextToken();
}

static long readLong() throws IOException {
    return Long.parseLong(next());
}

static int readInt() throws IOException {
    return Integer.parseInt(next());
}

static double readDouble() throws IOException {
    return Double.parseDouble(next());
}

static String readLine() throws IOException {
    return br.readLine().trim();
}
```

Typing Speed!

- Improve your typing speed!
- Be able to type without looking at keyboard
- Be able to type with all fingers

Understand the Problem!

- Read the problem carefully
- Read the problem several times
- Look for variables that you need
- Solve the sample test case by yourself
- Look to see if something needs to be specifically named (e.g. input/output files, Java class name)

Look at the Limits!

- Time limit
  - Time limit can be per **test case** or **entire execution time**
  - Time limit is **NOT** how long it takes you to code

Execution Results

Test case #1: TLE [>2.000s, 776.00 KB] (0/25)
Test case #2: TLE [>2.000s, 780.00 KB] (0/25)
Test case #3: TLE [>2.000s, 784.00 KB] (0/25)
Test case #4: TLE [>2.000s, 780.00 KB] (0/25)

"Ah, you bricked the server"

Look at the Limits!

- Memory limit
  - Be careful not to declare too many variables
  - Example: Array of 1 trillion integers
  - Remember sizes of primitive data types
  - Remember how many bytes in a megabyte, kilobyte, etc.

Execution Results

Test case #1: MLE [0.003s, 0K] (0/20) (Details)
Test case #2: MLE [0.003s, 0K] (0/20) (Details)
Test case #3: MLE [0.003s, 0K] (0/20) (Details)
Test case #4: MLE [0.003s, 0K] (0/20) (Details)
Test case #5: MLE [0.003s, 0K] (0/20) (Details)
Test case #6: MLE [0.000s, 0K] (0/100) (Details)

Your final score for this attempt is: 0/200

Look at the Limits!

- Variable limits
  - Almost all variables will have a limit
  - Example: $(1 \leq N \leq 1{,}000)$
  - **Not all variables affect running speed**

Constraints

For all subtasks:
$0 \leq A[i] \leq 10^5$ for all valid i.
$1 \leq l \leq r \leq N$
$1 \leq x \leq 10^5$

Subtask 1 [15%]

$M = 2$
$1 \leq N, Q \leq 1\,000$

Subtask 2 [15%]

$M = 2$
$1 \leq N, Q \leq 100\,000$

Subtask 3 [15%]

$M = 3$
$1 \leq N, Q \leq 100\,000$

Subtask 4 [15%]

$M = 5$
$1 \leq N, Q \leq 100\,000$

Subtask 5 [40%]

$M = 10\,007$
$1 \leq N, Q \leq 200\,000$

Assume Test Cases are Correct!

- Do not check if variables exceed limits
- Do not check if test cases are valid
- Checking wastes precious coding time
- Example:
  - In a problem, $(1 \leq N \leq 1{,}000)$
  - DO NOT check if N is in between 1 and 1,000, inclusive.
- If test cases are invalid, it is the problemsetter's fault

Make Your Variables Public!

- **Only do this in contests**
- Make important variables above your main code
- Easy for other methods to access variables
- Statically declare array sizes (use variable limit, plus a bit extra)
  - Example: $(1 \leq N \leq 1{,}000)$
  - Declare: `int array[1005];`

```
int N,X,Q;
ll qu,dist[3005];
pll points[3005];
bool vis[3005];

ll getDist(ll n1, ll n2){
    return n1*n1+n2*n2;
}

int main() {
    memset(dist,0x3f,sizeof(dist));
    scanf("%d",&N);
```

Use the Right Variable Size!

- Ensure that your variables will not **overflow**
- Limit of `int` in C++ is $[-2^{31}, 2^{31}-1]$
  - **About 2 billion**
  - If you exceed this interval, you **will not** get correct answers
- Limit of `long long` in C++ is $[-2^{63}, 2^{63}-1]$
  - **About 9 quintillion**
  - Calculations will generally not exceed this interval
- In C++, use `unsigned` to double the positive range, but lose ability to represent negatives
- Caution! `long` in C++ can be either 32 or 64 bit, so avoid using it

Partial Points!

- Some contests give you part marks for partially correct solutions (including CCC)
- If you cannot solve the entire problem, try to earn partial points
- Look for **subtasks** in the problem
- Easier to solve test cases with smaller input sizes

Constraints

Subtask 1 [20%]

$1 \leq N \leq 100, 0 \leq x_i, y_i \leq 100$

$1 \leq Q \leq 10, 0 \leq Q_i \leq 10^5$

Subtask 2 [30%]

$1 \leq N \leq 1000, 0 \leq x_i, y_i \leq 10^4$

$1 \leq Q \leq 1000, 0 \leq Q_i \leq 10^9$

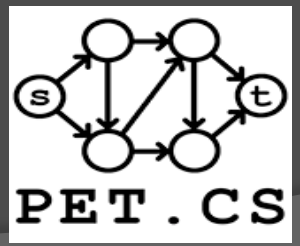
Subtask 3 [50%]

$1 \leq N \leq 3000, 0 \leq x_i, y_i \leq 10^6$

$1 \leq Q \leq 10^6, 0 \leq Q_i \leq 10^{14}$

Practice Practice Practice!

- Practice!
- Try past problems from the same contest series you are training for
- Do additional contests!



Efficiency

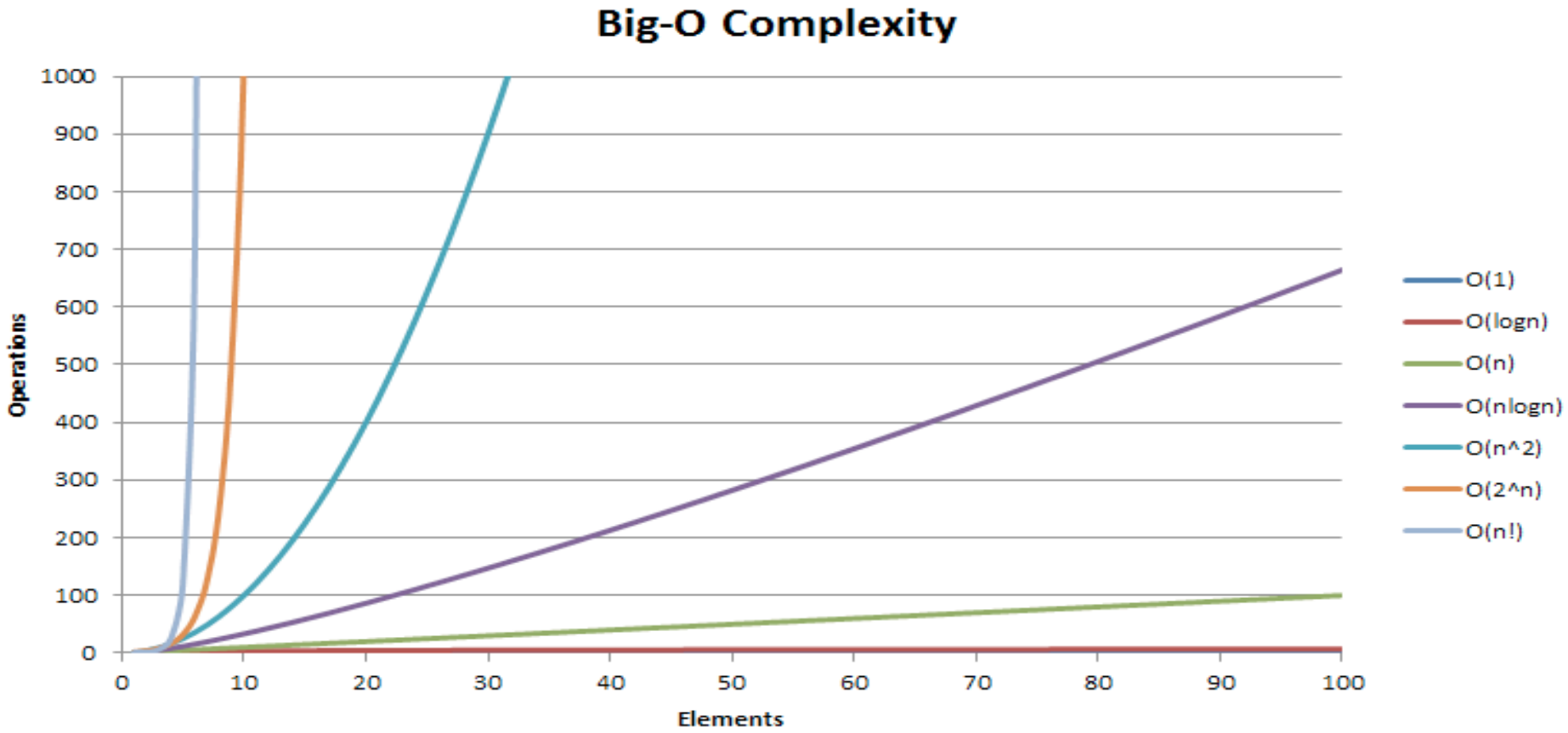
Algorithm Analysis

- Determination of speed/memory usage of algorithms (solutions to problems)
- Analysis of speed is called **time complexity**
- Usually, pay more attention to time complexity of algorithm rather than memory usage
  - Memory limits are usually more lenient

Big O Notation

- **Big O Notation** is frequently used
  - Asymptotic Notation
- Shows how a function (the algorithm) grows (time and memory) as input becomes larger
- Describes **limiting behavior** in terms of simpler, parent functions
- To simplify a function, use the term with the highest order, without coefficients
  - e.g. $3N^3 + N^2 + 10$ is $O(N^3)$
- You do not need to use a fully simplified function all of the time, more information may be helpful

Growth of Functions

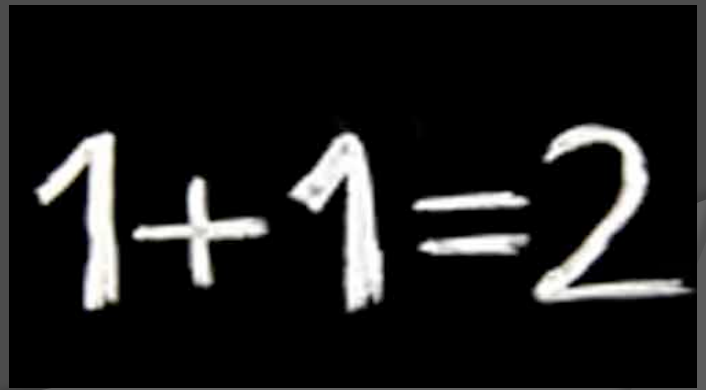


Efficiency

- Efficient algorithms are algorithms that do not need to use as many resources
- For contests, an efficient algorithm generally tries to run as fast as possible, then tries to conserve as much memory
- Time complexities and memory complexities with slowly growing big O notation functions are more efficient

O(1)

- **Constant**
- Adding two integers
- Getting size of a string

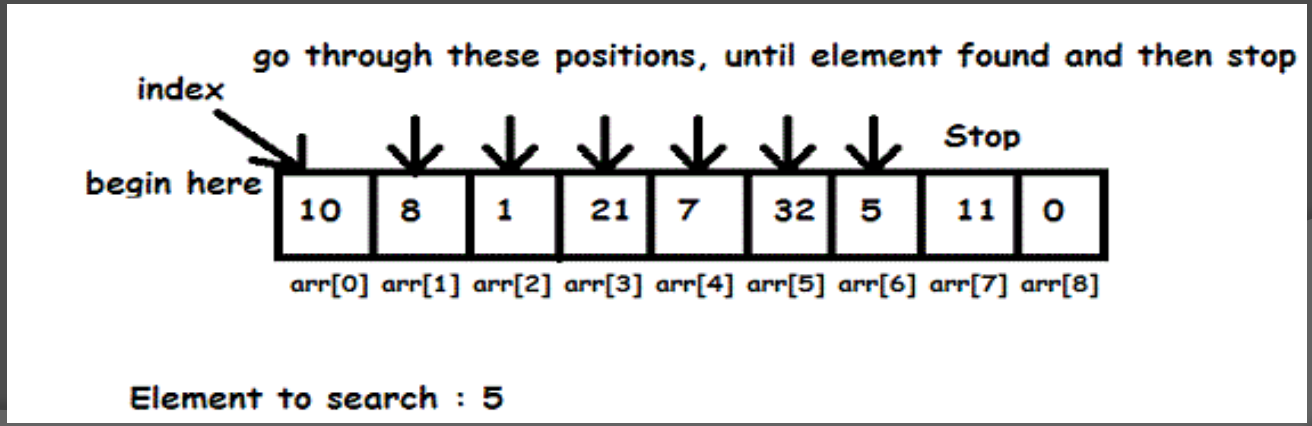


O(log N)

- **Logarithmic**
- **NOTE:** In computer science, common logarithm is to **base 2, NOT base 10**
- Binary Search
- Quick-Power algorithm

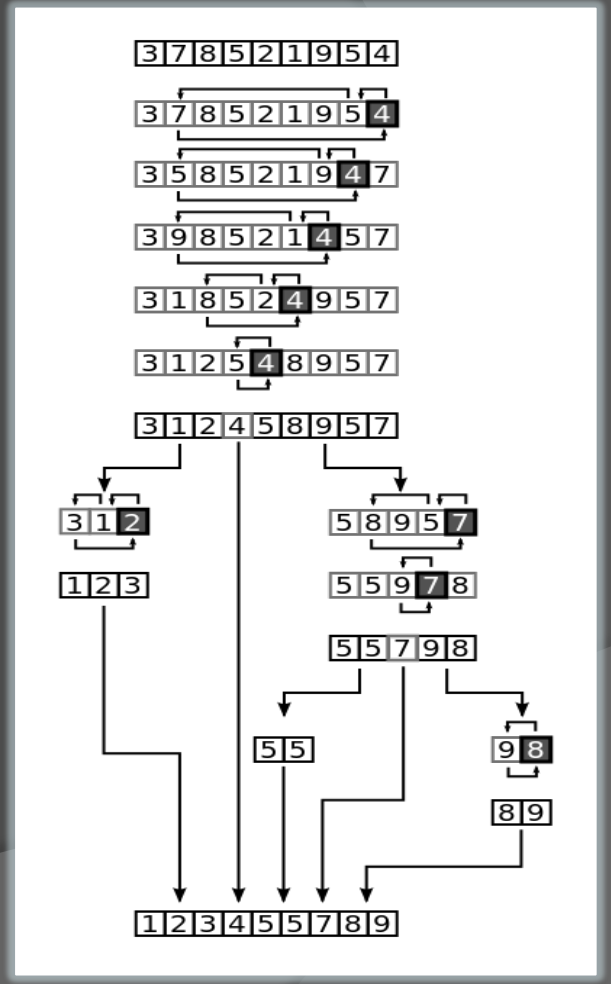
O(N)

- **Linear**
- Regular searching through a 1-D array
- Getting a substring of a string



O(N log N)

- Also seen as *(N*log N)*
- **Linearithmic**
- Best sorting algorithm
- Binary searching N times



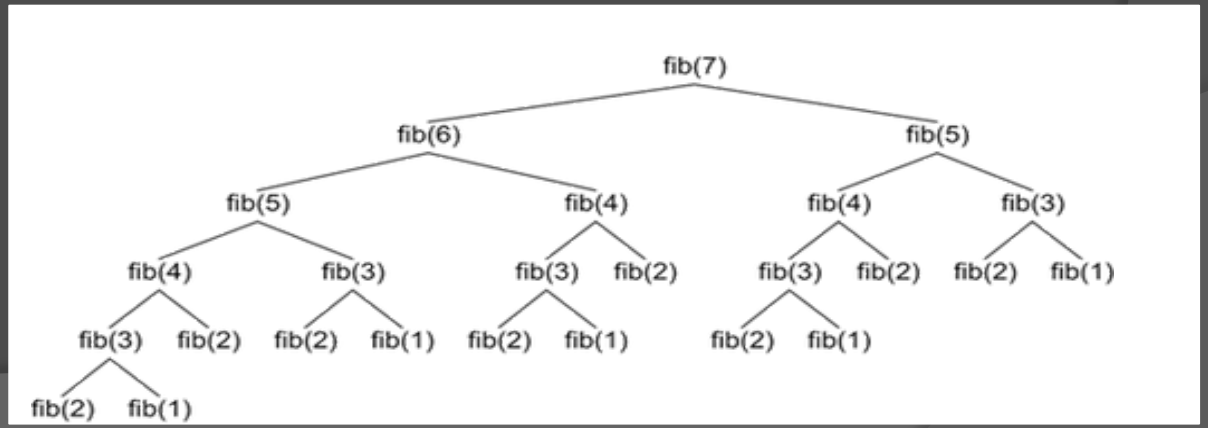
$O(N^2)$ or $O(N^k)$

- **Quadratic or Polynomial**
- Searching through a 2-D array (N by N)
- Finding all possible pairs in a list of integers

a[0][0]	a[0][1]	a[0][2]	a[0][3]
a[1][0]	a[1][1]	a[1][2]	a[1][3]
a[2][0]	a[2][1]	a[2][2]	a[2][3]

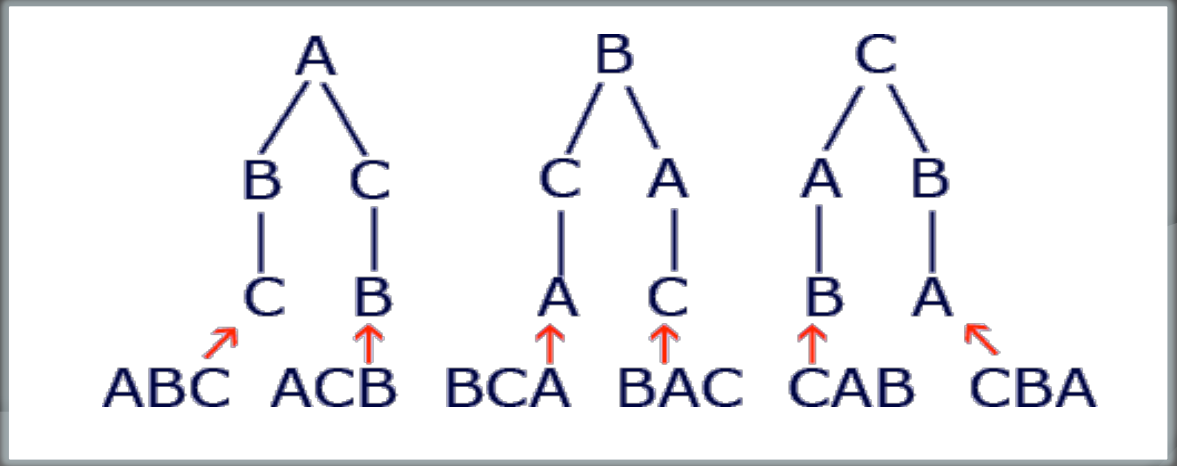
$O(2^N)$

- **Exponential**
- Recursive Fibonacci function without storing answers
- Finding all possible combinations of a list
  - **Order DOES NOT matter**, (1,2,3) is the same as (3,2,1)



O(N!)

- **Factorial**
- Finding all possible permutations of a list
  - **Order matters**, (1,2,3) is different from (3,2,1)



Even worse

- $O(N^N)$
- $O(N^N * N!)$
- $O(\infty)$

Big O Notation Practice

- Practice: What is the asymptotic time bound of functions with these time complexities?
- $3N + 2N + N$ $O(N)$
- $2N^3 + 5N^2$ $O(N^3)$
- $N + \log N$ $O(N)$
- $N^2 \log N$ $O(N^2 \log N)$
- $2^N + N^2$ $O(2^N)$
- 10 $O(1)$

Finding Time Complexity

- Need to know how the algorithm runs – its process
- Find a relationship between number of constant operations and input size
- Memorize complexity for common algorithms (sorting, iterating, binary searching, Dijkstra, etc.)
- Example: 1 nested for loop requires N^2 operations
- Based on experience

Finding Time Complexity

What is the time complexity of this code?

```
int ar[1005],N;
//get input
int maxVal = ar[0];
for (int i = 1; i < N; i++){
        maxVal = max(maxVal,ar[i]);
}
printf("%d\n",maxVal);
```

Time complexity: $O(N)$

Finding Time Complexity

```
int ar[1005][1005],N;
//get input
for (int i = 0; i < N; i++){
        sort(ar[i],ar[i]+N);
}
```

Time complexity: $O(N^2 \log N)$

Finding Time Complexity

```
int ar[1005][1005],N;
//get input
for (int i = 0; i < N; i++){
        for (int j = 0; j < N/2; j++){
                swap(ar[i][j],ar[i][N-j]);
        }
}
```

Time complexity: $O(N^2)$

Finding Time Complexity

```
string func(string str, int idx, int len){
        if (idx == len) cout << str << endl;
        else
                func(str+"a",idx+1,len), func(str+"b",idx+1,len);
}

int N;
//get input
func("",0,(int)log2(N));
```

Time complexity: $O(N \log N)$

(printing each string is $O(\log N)$, $O(N)$ strings are formed)

Finding Time Complexity

```
int N;
//get input
vector<bool> bin
while(N){
        bin.push_back(N%2);
        N /= 2;
}
```

Time complexity: $O(\log N)$

Finding Time Complexity

```
int N; pdd points[100005]; //pdd is pair<double,double>
//get input
vector<pdd> hull(2*N);
int k = 0;
sort(points,points+N);
for (int i = 0; i < N; i++){
        //cross() is O(1)
        while(k >= 2 && cross(hull[k-2],hull[k-1],points[i]) <= 0) k--;
        hull[k++] = points[i];
}
for (int i = N-2, t = k+1; i >= 0; i--){
        while(k >= t && cross(hull[k-2],hull[k-1],points[i]) <= 0) k--;
        hull[k++] = points[i];
}
```

Time complexity: $O(N \log N)$

Relating Time Complexity to Time Limits

- How can knowing time complexity help?
  - Hint: **C++ can do approximately 20 million operations per second on online judges**
  - By looking at the time limit and variable limits, you can estimate time complexity of the correct algorithm
  - **Remember, not all variables affect the time complexity of the algorithm**

Determining Approximate Running Time

- Given a time complexity in big O notation:
  - Set all variables in the function equal to the limits given in the problem
  - Evaluate the function
  - The algorithm will take approximately 1 second for every 20,000,000 – 50,000,000
- e.g. $O(MN^2),\ 1 \leq M \leq 100,\ 1 \leq N \leq 2,000$

  MN^2

  $= (100)*(2,000)^2$

  $= 400,000,000$

  Too slow to pass if time limit is not high!

Estimating Time Complexities

- The time limit and variable limits of the problem are given
- Guess some function using big O notation
- Evaluate the function using the limits of the variable
- Compare the result with the time limit

Estimating Time Complexities

- Time limit: 1s
- Variable limit: $1 \leq N \leq 10$
- Brute force algorithm
- Factorial time algorithm
- Exponential time algorithm

Estimating Time Complexities

- Time limit: 2s
- Variable limit: $1 \leq N \leq 300$
- Cubic time algorithm

Estimating Time Complexities

- Time limit: 2s
- Variable limit: $1 \leq N \leq 1,000$
- Quadratic time algorithm

Estimating Time Complexities

- Time limit: 1s
- Variable limit: $1 \leq N \leq 10,000$
- Linearithmic time algorithm

Estimating Time Complexities

- Time limit: 10s
- Variable limit: $1 \leq N \leq 10,000$
- Quadratic time algorithm

Estimating Time Complexities

- Time limit: 2s
- Variable limit: $1 \leq N \leq 100,000$
- Linearithmic time algorithm

Estimating Time Complexities

- Time limit: 2s
- Variable limit: $1 \leq N \leq 1,000,000$
- Linearithmic time algorithm
- Linear time algorithm

Estimating Time Complexities

- Time limit: 2s
- Variable limit: $1 \leq N \leq 1,000,000,000$
- Logarithmic time algorithm
- Constant time algorithm

Estimating Time Complexities

- Time limit: 2s
- Variable limit: $1 \leq N \leq 10^{100}$
- Ignore this variable, it is unnecessary

THANK YOU!