CS 211 : Lecture 11

- **High-level: Data structures and algorithms**
  - Example: search and sort
- **Data structure: Linked List**
  - Live code a simple map using a linked list
- **Project 04 data structure**

Northwestern University

Recall Project 02, nodes.find()

- **We gave you code for "binary search" to replace the "linear search" implementation…**

How to reduce the search cost? We need a better algorithm: **binary search**. Comment out the linear search code in the ***find( )*** function of the ***Nodes*** class, and replace with binary search. It's a beautiful algorithm (watch a visualization here). Note that binary search requires the search data to be in sorted order --- luckily this is already true because the OSM files provide the node elements in ascending order. Here's the code for binary search, which you'll learn more about in CS 214:

```
//
// binary search: jump in the middle, and if not found, search to
// the left if the element is smaller or to the right if bigger.
//
int low = 0;
int high = (int)this->osmNodes.size() - 1;

while (low <= high) {
  int mid = low + ((high - low) / 2);

  long long nodeid = this->osmNodes[mid].getID();

  if (id == nodeid) { // found!
    lat = this->osmNodes[mid].getLat();
    lon = this->osmNodes[mid].getLon();
    isEntrance = this->osmNodes[mid].getIsEntrance();

    return true;
  }
  else if (id < nodeid) { // search left:
    high = mid - 1;
  }
  else { // search right:
    low = mid + 1;
  }
}//while
```

But It's All Just Code…Right?

- **Not really! Software Quality includes:**
  - *Correctness*
  - *Efficiency*
    - How fast
    - How much memory
    - Energy consumption
  - *Flexibility*
  - *Maintainability*
  - *Etc.*

Interested in seeing ALL Software Quality Attributes?
https://iso25000.com/index.php/en/iso-25000-standards/iso-25010

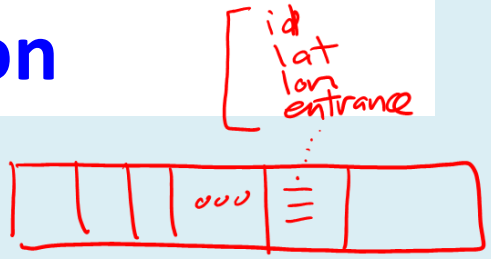
But It's All Just Code…Right?

- **Not really! Software Quality includes:**
  - *Correctness*
  - ***Efficiency***
    - **How fast**
    - **How much memory**
    - Energy consumption
  - *Flexibility*
  - *Maintainability*
  - *Etc.*

CS214: Data Structures and Algorithms ☺

Interested in seeing ALL Software Quality Attributes?
https://iso25000.com/index.php/en/iso-25000-standards/iso-25010

Let's Check Your Intuition



- **Assume a vector contains 15,000 elements, and we are using linear search.**

- **To output "University Hall", the program has to lookup 24 elements in the vector to obtain position information. If the cost of accessing an element in the vector is $1, how much does it cost on average to lookup these 24 elements?**

```
Enter building name (partial or complete), or * to list, or $ to end>
University Hall
University Hall
Address: 1897 Sheridan Road
Building ID: 33908928
Nodes: 24
 388499432: (42.0518, -87.6758)
 4774714375: (42.0518, -87.6758)
 2241369266: (42.0518, -87.6758)
 2241369264: (42.0518, -87.6759)
 2241227052: (42.0519, -87.6758), is entrance
 4774714382: (42.0519, -87.6758)
 4774714383: (42.0519, -87.6758)
 388499433: (42.052, -87.6758)
 388499434: (42.0521, -87.676)
 1766764521: (42.0519, -87.6761), is entrance
 4774714381: (42.0519, -87.6761)
 4774714380: (42.0519, -87.6761)
 388499436: (42.0518, -87.6762)
 4774714372: (42.0518, -87.676)
 2241226778: (42.0518, -87.676)
 2241227054: (42.0518, -87.676), is entrance
 2241226814: (42.0517, -87.676)
 4774714373: (42.0518, -87.676)
 4774714374: (42.0517, -87.6759)
 4774714376: (42.0517, -87.6759)
 4774714377: (42.0517, -87.6758)
 4774714379: (42.0517, -87.6758)
 4774714378: (42.0517, -87.6758)
 388499432: (42.0518, -87.6758)
```

A) $300

B) $24,000

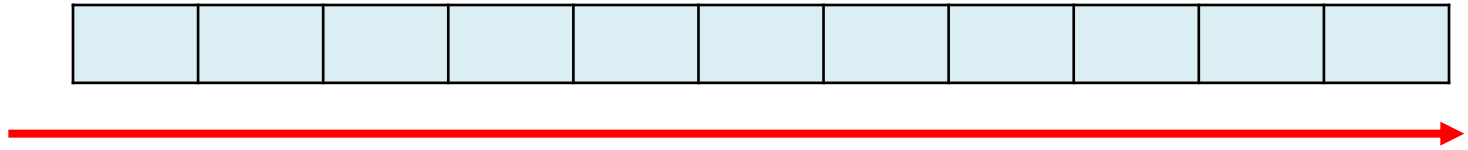
C) $180,000

D) $359,700

E) Over $1,000,000

Discussion

- **Linear search**



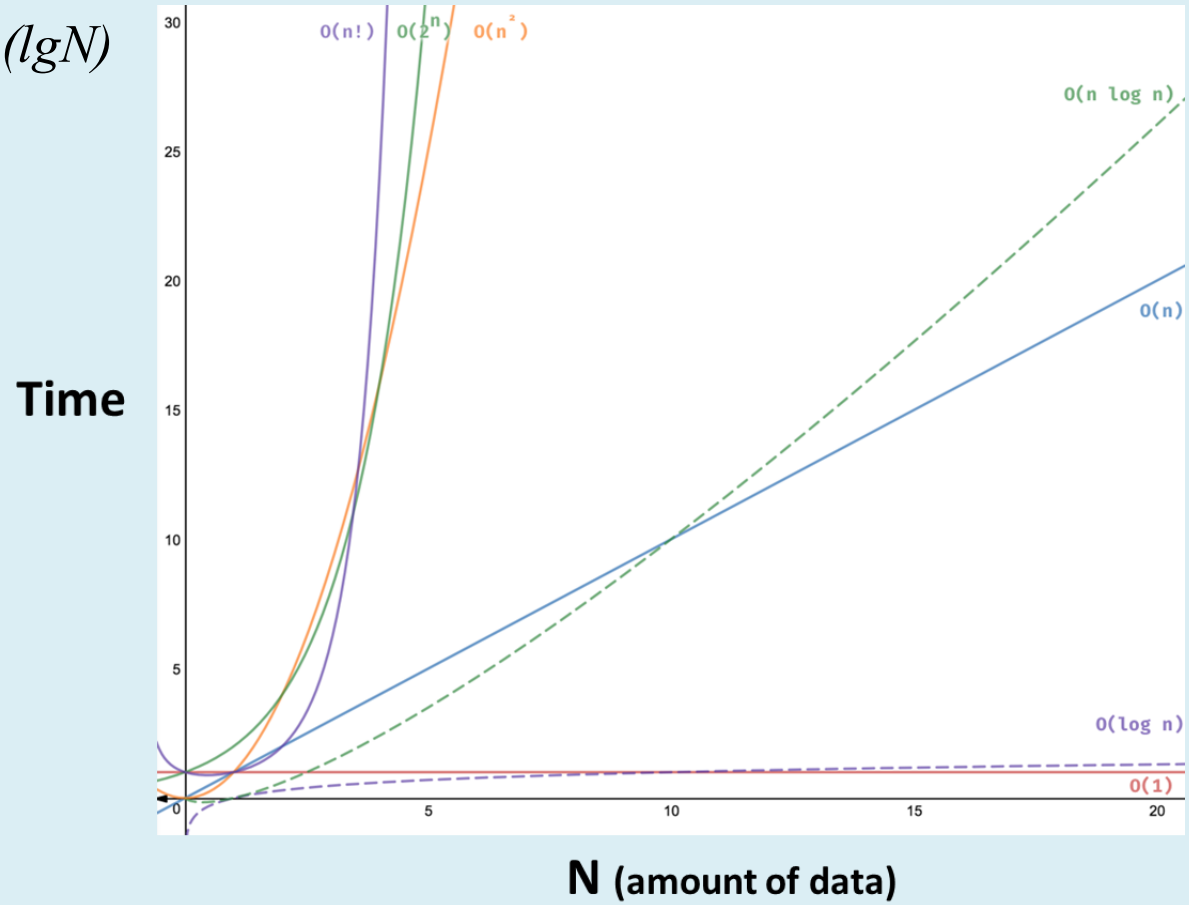
- **Average cost?**
  - 24 * cost of searching ½ the vector
  - 24 * 7,500
  - $180,000
- **We say the linear search algorithm has a time complexity "on the order of N"**
  - *as N increases, the time increases*
  - *Written O(N)* ***(not assessed in CS211)***

Binary search

2	4	9	14	18	22	36	54	71	88	101

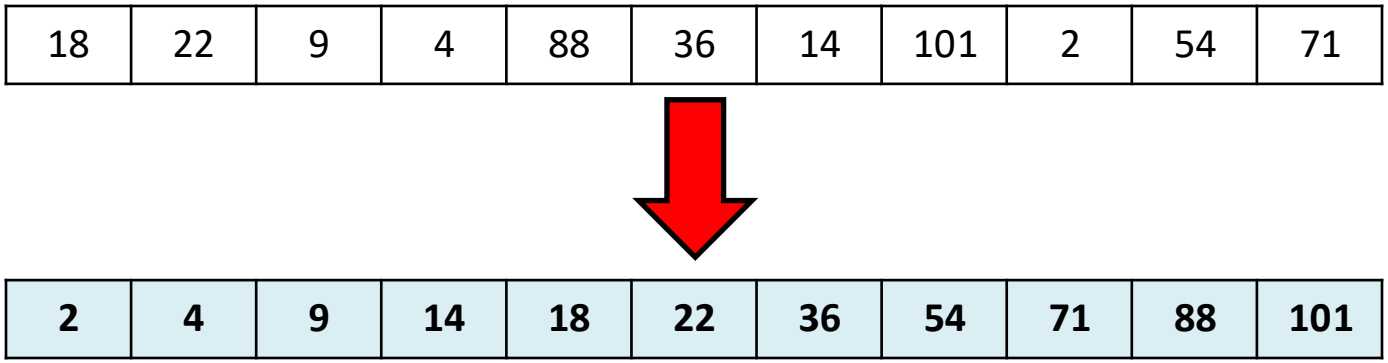
1. Jump to the middle
2. Compare --- if == stop, if < go left, if > go right
3. Repeat

- **Binary search is a *divide-and-conquer algorithm*, dividing the search space in half each time…**
- **We say the binary search algorithm has a time complexity "on the order of $\log_2 N$"**
  - *Written O(lgN)*



Binary search pre-condition: sorted order

- **To use binary search, data must be sorted!**

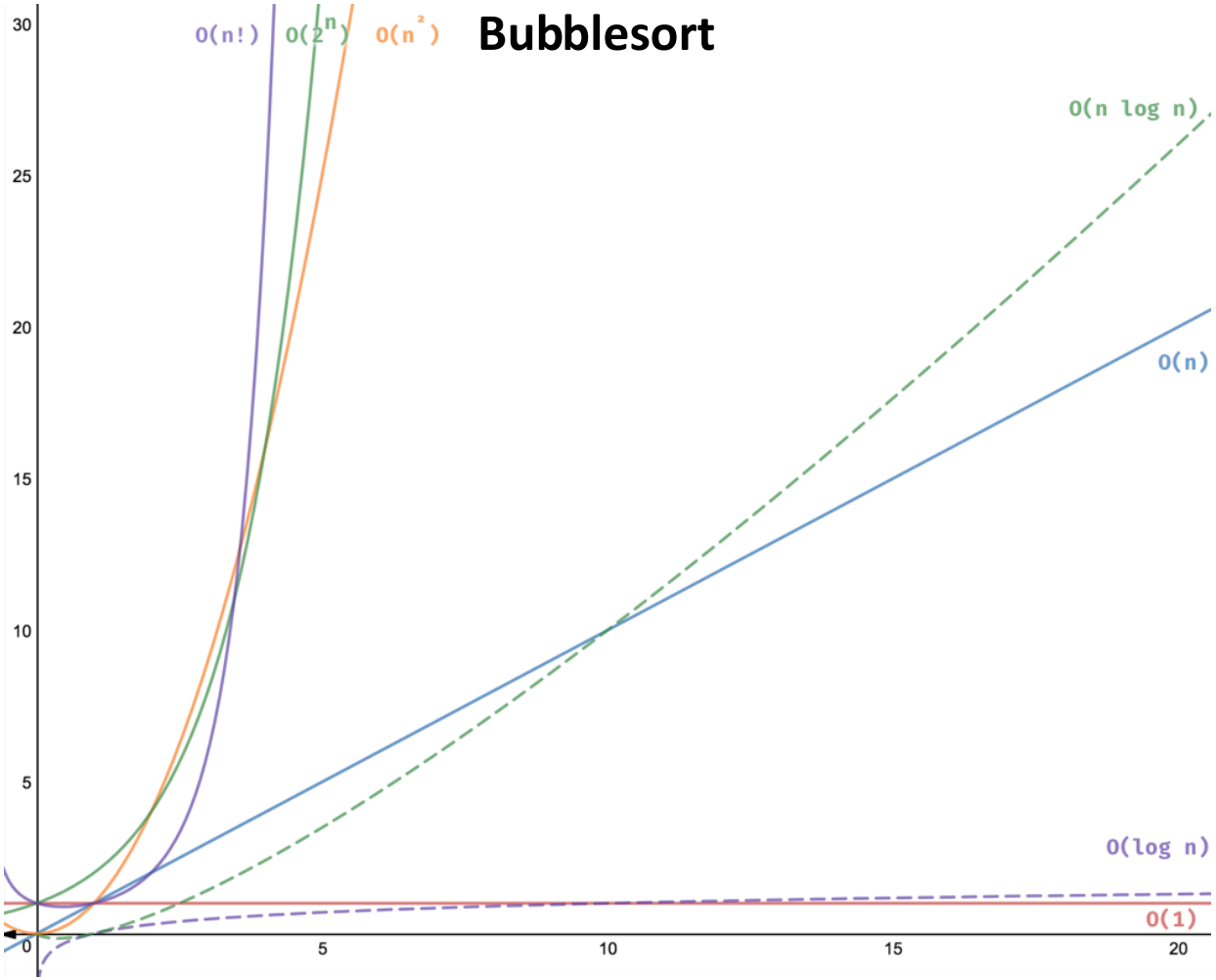


Sorting Algorithms

Sorting visualizations:

- http://www.sorting-algorithms.com
- http://www.cs.usfca.edu/~galles/visualization/ComparisonSort.html
- **"15 sorts in 6 minutes" video on YouTube:** https://www.youtube.com/watch?v=kPRA0W1kECg

Sort: Time Complexities



Bubblesort

Mergesort

Linear search

Binary search

Time

N (amount of data)

Is efficient algorithm alone enough?

- **Smart design and choice of Data Structures matter too (again, more in CS214☺)**
- **Data structures we have used in CS211:**
  - *Vector*
  - *Array*

Today's Plan

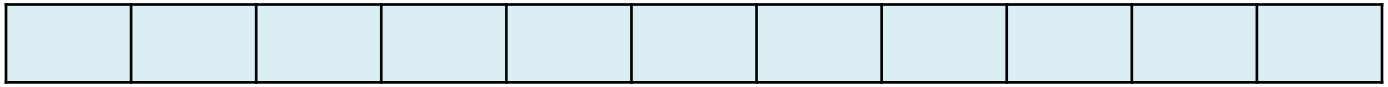
1. **Introduce *Linked List* in the context of a simple Map implementation**
2. **Code the simple Map together**
3. **Explain Project 04**

Next time: introduce how C++ Map is implemented (which is built upon linked list, but much fancier ☺); **other cool things associated with Map like Iterator**

What we know: Vector and Array

Pro: Accessing an element at index i is very efficient!
(we call it "constant time", O(1))

Con: Inserting/Deleting in the middle requires shifting elements



High level: Why Linked List?

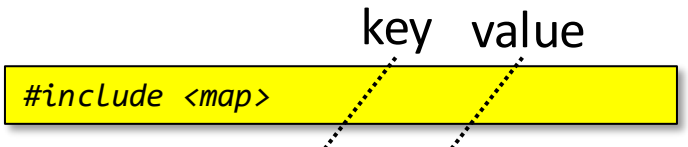
In some cases, we want to reverse the advantage and disadvantage of vector and array --

Pro: Inserting/Deleting in the middle is efficient

Con: Accessing an element at index i is not efficient

Recall: map in C++

- **map** is an abstraction




```
int main()
{
  map<string, int>  animals;

  animals["elephant"] = 1;
  animals["cat"] = 3;
  animals["owl"] = 1;
  animals["dog"] = 2;
  .
  .
  .
  string type;
  cout << "Enter a type of animal> ";
  cin >> type;

  cout << "I own " << animals[type] << " animals of this type" << endl;
```

Recall: Traversing a Map

- use *foreach*

```
int main()
{
   map<string, int>  animals;
   .
   .
   .

   cout << "Animals I own:" << endl;

   for (pair<string,int> kv_pair : animals)  // for each pair:
     cout << kv_pair.first << ": " << kv_pair.second << endl;
```

```
Animals I own:
bear: 1
cat: 3
dog: 2
elephant: 1
ferret: 6
moth: 9
owl: 1
pig: 4
zebra: 1
```

C++ Map: Order of the Elements

- **C++ built-in map seems to store elements in *some order***

```
int main()
{
   map<string, int>  animals;
   .
   .
   .

   cout << "Animals I own:" << endl;

   for (pair<string,int> kv_pair : animals)  // for each pair:
     cout << kv_pair.first << ": " << kv_pair.second << endl;
```

```
Animals I own:
bear: 1
cat: 3
dog: 2
elephant: 1
ferret: 6
moth: 9
owl: 1
pig: 4
zebra: 1
```

Let's Build a Simple Map

- **The simple map is built using a linked list**
- **But it won't store elements in the same order as how C++ map does it yet**
  - *Elements will be ordered in the last in first out order*
- **You will write more features upon this simple map in Lab 05**
- **The idea of a linked list and traversing a linked list is used heavily in Project 04**

Live Coding

- **Please setup: onlinegdb or VSCode**
  - *Create an empty file: main.cpp*

The complete code will be posted on Canvas after class.

Take a Break! 5min

- **Stretch, get water, take a deep breath, yawn, stand up, walk around etc. whatever you need to do ☺**

Summary: What is a Linked List?

- **Storing the elements individually in separate pieces of memory, linking them together with pointers**
- **Each element of a linked-list is a *NODE* object that contains at least 2 data members: the data and a pointer to the next element.**

Project 04: Data Structure

- **You were given the code.**

```
struct STMT
{
//
// what kind of stmt do we have?
//
int stmt_type; // enum STMT_TYPES
int line; // what line # does it start on?

//
// pointer to that stmt struct:
//
union
{
struct STMT_ASSIGNMENT* assignment;
struct STMT_FUNCTION_CALL* function_call;
struct STMT_IF_THEN_ELSE* if_then_else;
struct STMT_WHILE_LOOP* while_loop;
struct STMT_PASS* pass;
} types;
};
```

```
struct STMT_ASSIGNMENT
{
//
// Examples: x = 123
// *p = x + y
//
char* var_name;
bool isPtrDeref;
struct VALUE* rhs;

struct STMT* next_stmt;
};
```

Project 04 Data Structure: Visualization

```
struct STMT
{
int stmt_type; // enum STMT_TYPES
int line; // what line # does it start on?
union
{
struct STMT_ASSIGNMENT* assignment;
struct STMT_FUNCTION_CALL* function_call;
struct STMT_IF_THEN_ELSE* if_then_else;
struct STMT_WHILE_LOOP* while_loop;
struct STMT_PASS* pass;
} types;
};
```

```
struct STMT_ASSIGNMENT
{
char* var_name;
bool isPtrDeref;
struct VALUE* rhs;

struct STMT* next_stmt;
};
```

Almost Done!

- **In-class Handout: clear and muddy (2min)**

Before You Leave

- **Submit your in-class handout**
  - **Make sure it has your name and NetID**
  - **Drop it in the box by the door(s)**

- **That's it! See you on Thursday!**