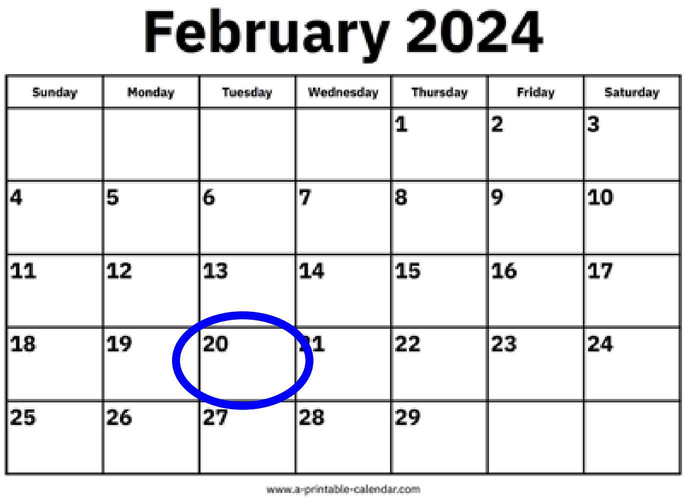
CS 211 : Tues 02/20 (lecture 14)

- **Topics**: searching, sorting, map

Prof. Hummel
(he/him)



February 2024

Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29		

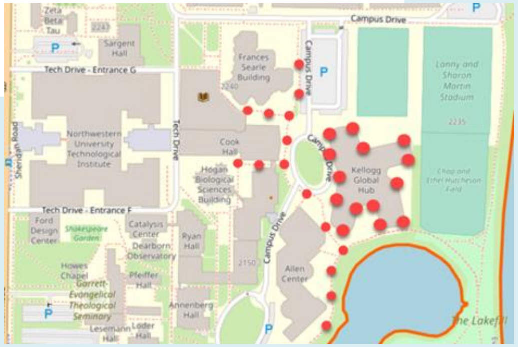
www.a-printable-calendar.com

Notes:

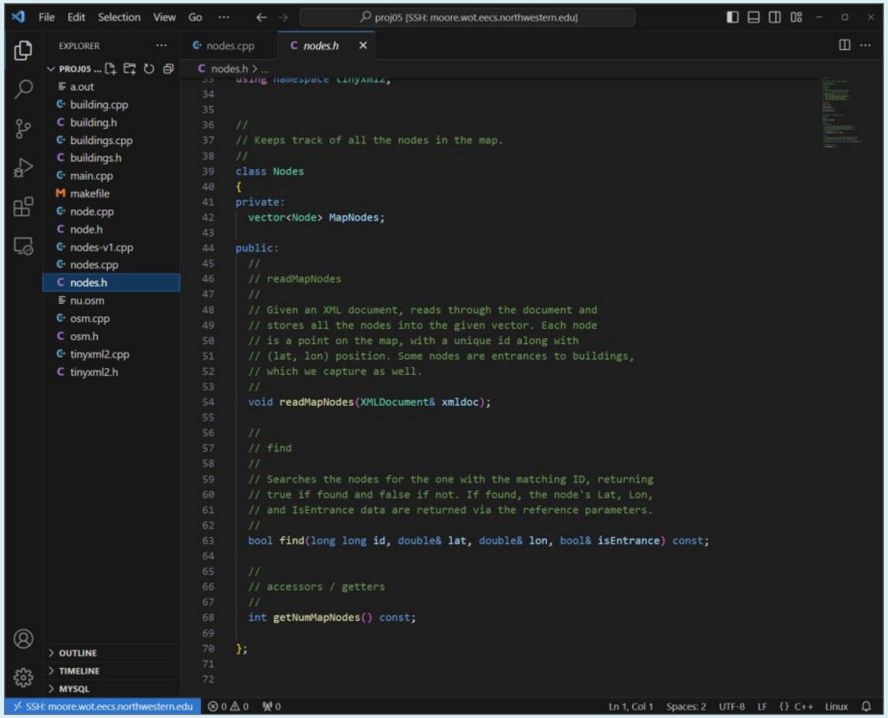
- *Lecture slides available on Canvas*
- ***No class Thursday!***
- ***HW 06** due tonight (Tuesday)*
- ***Project 06** due Friday night (can submit as late as Sunday with late days)*

Northwestern University

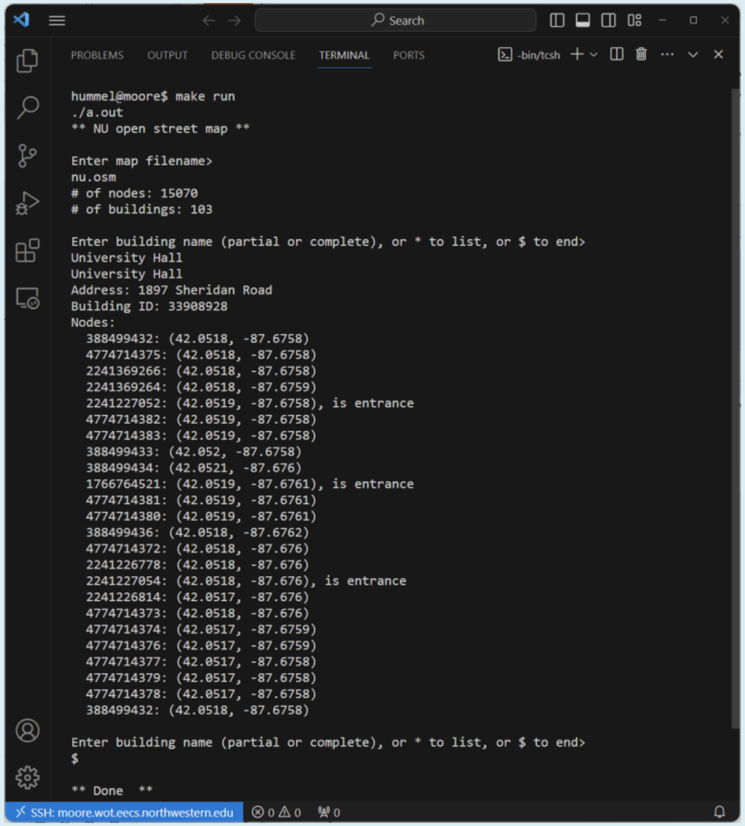
Projects 05 – 08



- **Working with Open Street Maps**
- **Working with EECS computers (aka Linux)**
  - *Recommendation: open two instances of VS code, editor + terminal*

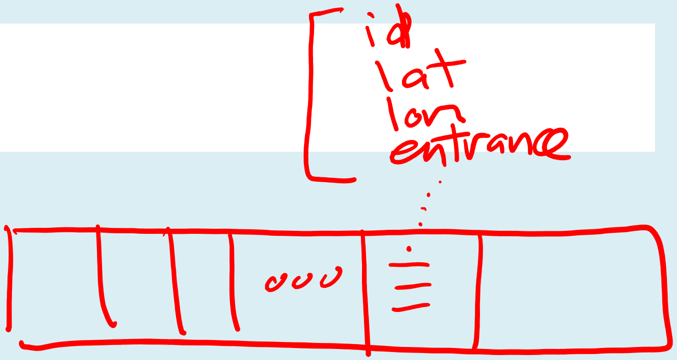


```cpp
using namespace tinyxml2;


//
// Keeps track of all the nodes in the map.
//
class Nodes
{
private:
  vector<Node> MapNodes;

public:
  //
  // readMapNodes
  //
  // Given an XML document, reads through the document and
  // stores all the nodes into the given vector. Each node
  // is a point on the map, with a unique id along with
  // (lat, lon) position. Some nodes are entrances to buildings,
  // which we capture as well.
  //
  void readMapNodes(XMLDocument& xmldoc);

  //
  // find
  //
  // Searches the nodes for the one with the matching ID, returning
  // true if found and false if not. If found, the node's Lat, Lon,
  // and IsEntrance data are returned via the reference parameters.
  //
  bool find(long long id, double& lat, double& lon, bool& isEntrance) const;

  //
  // accessors / getters
  //
  int getNumMapNodes() const;

};
```



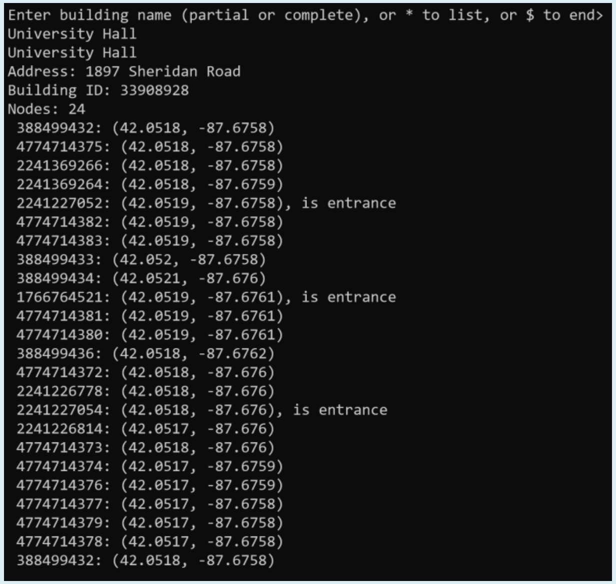
```
hummel@moore$ make run
./a.out
** NU open street map **

Enter map filename>
nu.osm
# of nodes: 15070
# of buildings: 103

Enter building name (partial or complete), or * to list, or $ to end>
University Hall
University Hall
Address: 1897 Sheridan Road
Building ID: 33908928
Nodes:
  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)

Enter building name (partial or complete), or * to list, or $ to end>
$

** Done  **
```

Nodes



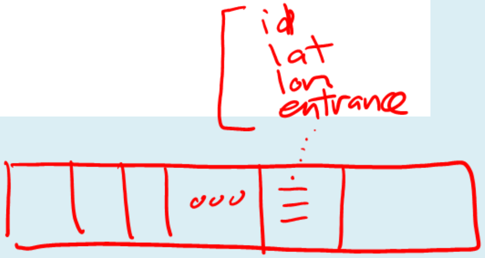
- For "nu.osm", there are 15,000 nodes (positions) in the Nodes vector:

- University Hall has 24 "node refs" outlining the perimeter of the building:



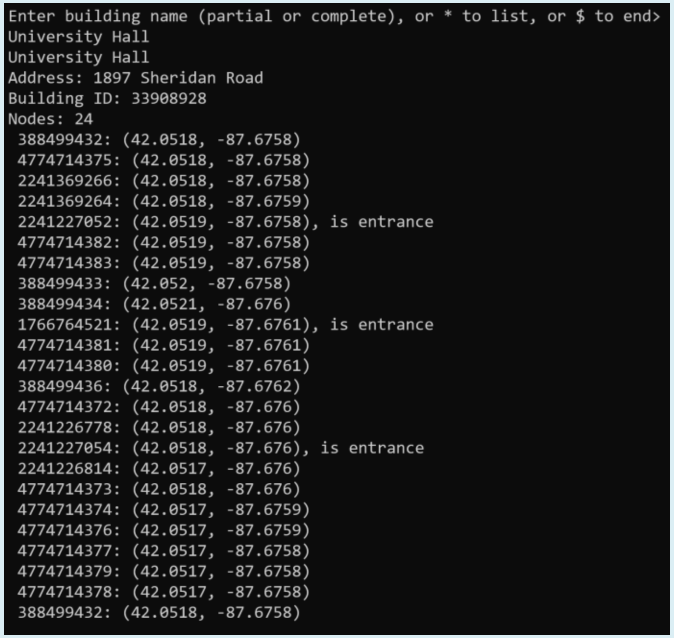
```
Enter building name (partial or complete), or * to list, or $ to end>
University Hall
University Hall
Address: 1897 Sheridan Road
Building ID: 33908928
Nodes: 24
 388499432: (42.0518, -87.6758)
 4774714375: (42.0518, -87.6758)
 2241369266: (42.0518, -87.6758)
 2241369264: (42.0518, -87.6759)
 2241227052: (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)
```

Question



- Assume linear search of the Nodes vector, which contains 15,000 elements.

- When outputting "University Hall", the program has to lookup these 24 references to obtain the lat/lon of each node. If the cost of accessing a node in the vector is $1, how much does it cost (on average) to lookup these 24 nodes?



```
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) $359,700

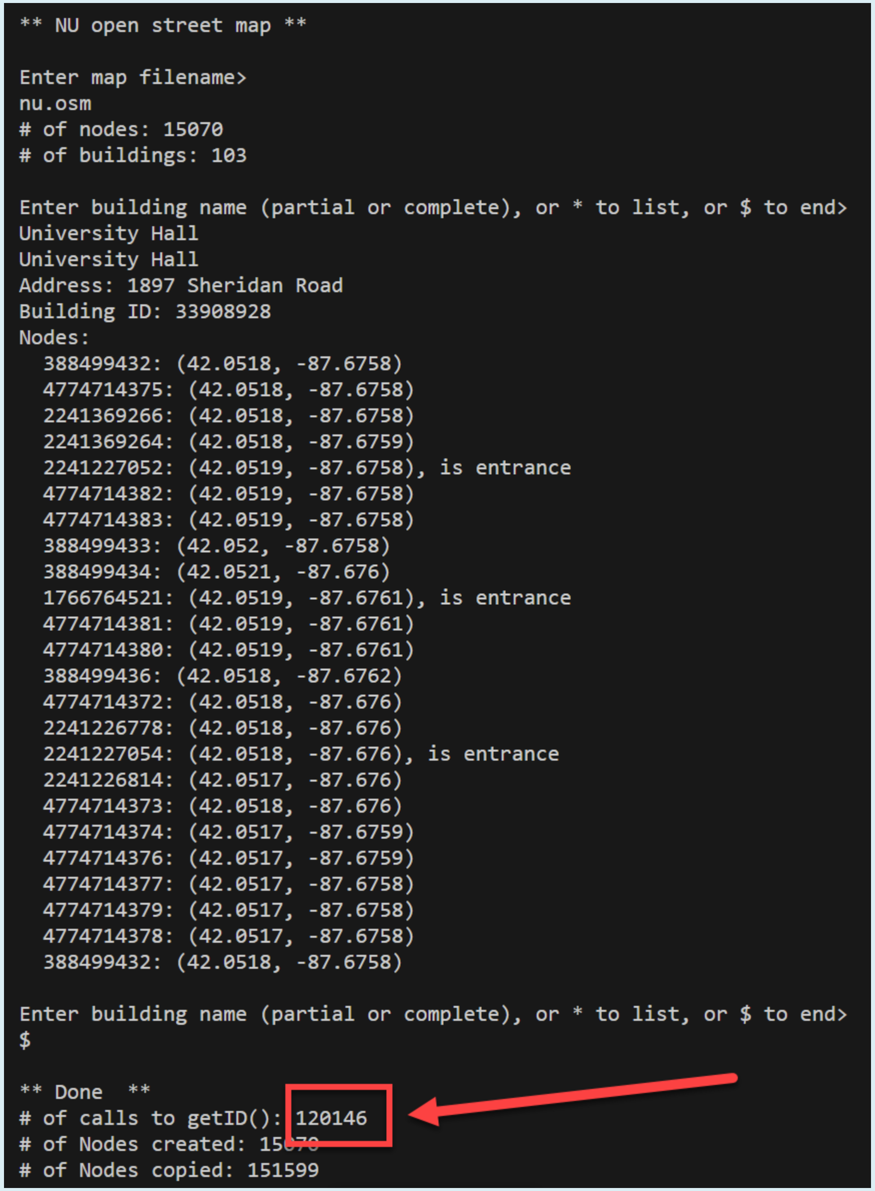
C) $180,000

D) $24,000

E) Over $1,000,000

Demo: linear search

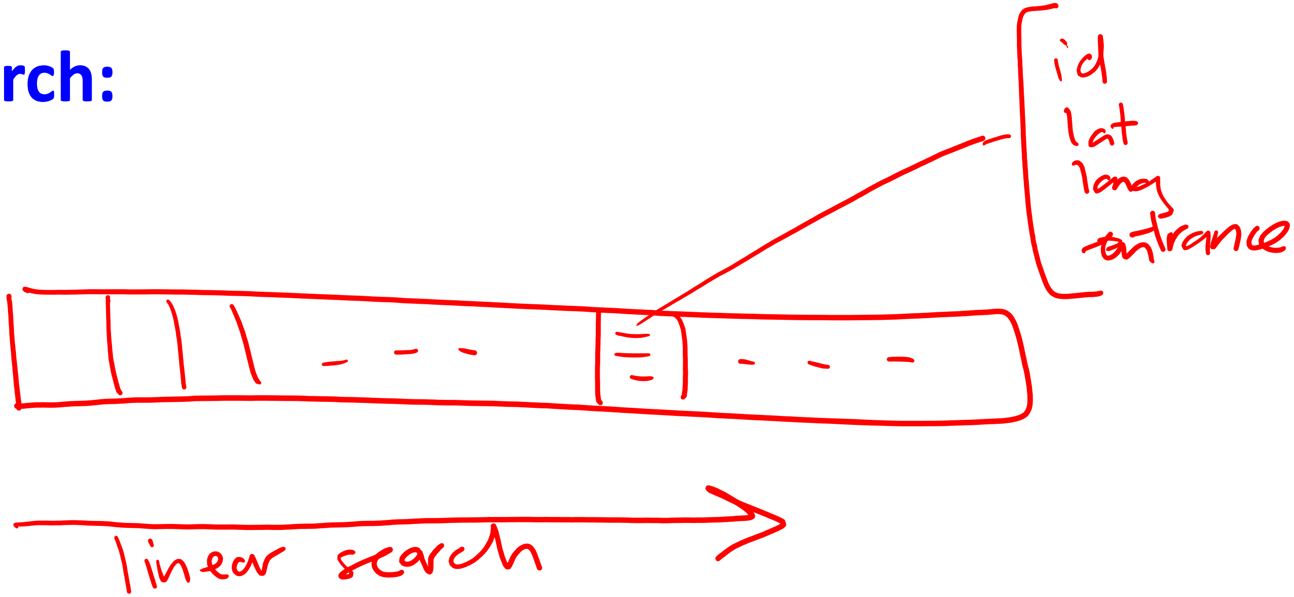
- **Let's confirm the result...**



```
** NU open street map **

Enter map filename>
nu.osm
# of nodes: 15070
# of buildings: 103

Enter building name (partial or complete), or * to list, or $ to end>
University Hall
University Hall
Address: 1897 Sheridan Road
Building ID: 33908928
Nodes:
  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)

Enter building name (partial or complete), or * to list, or $ to end>
$

** Done  **
# of calls to getID(): 120146
# of Nodes created: 15070
# of Nodes copied: 151599
```

Discussion

- **Linear search:**



- **Costs?**
  - *Best (lowest) possible cost: 1 + 2 + … + 24 = n(n+1)/2 = $300*
  - *Worst (highest) possible cost (none found): 24 * 15000 = $360,000*
  - *On average, you expect the cost to be: 24 * 7,500 = $180,000*

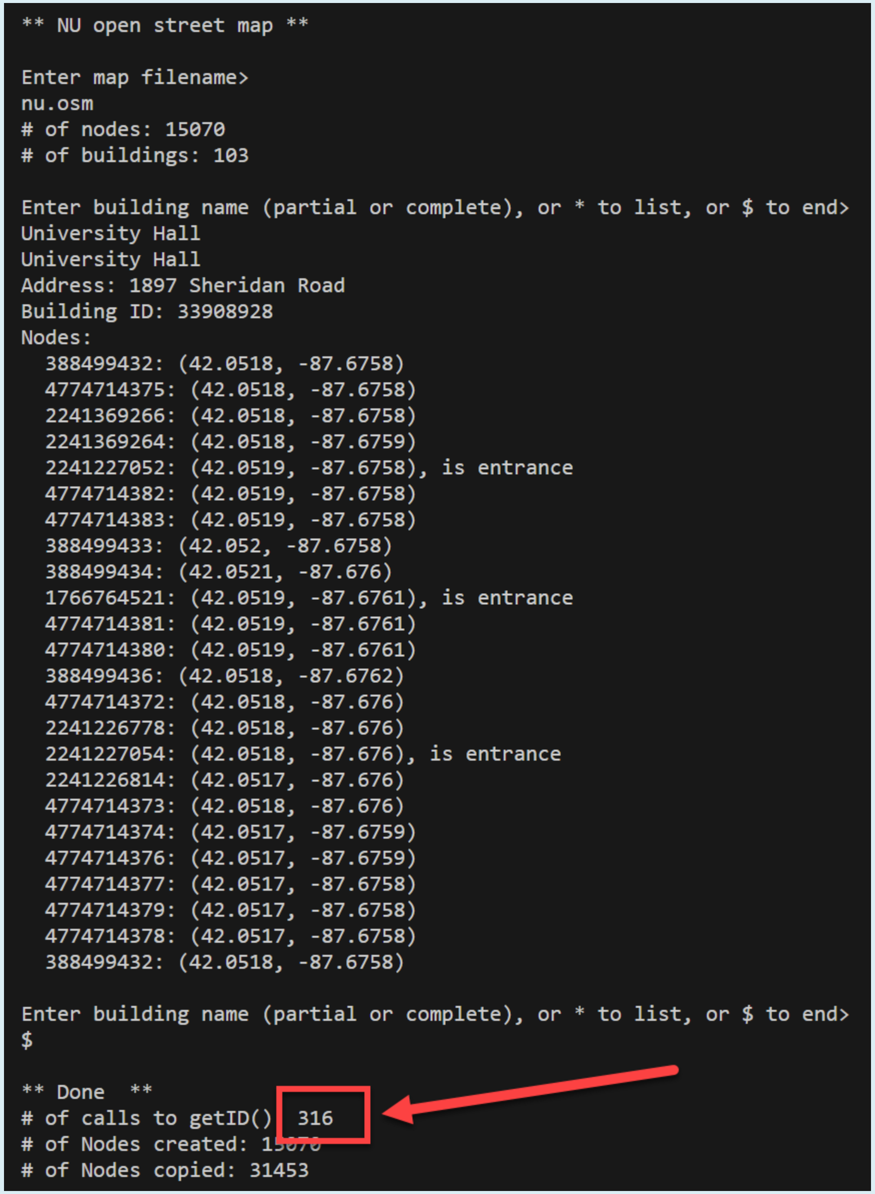
- **We say the linear search algorithm has a time complexity "on the order of N"**
  - *as N increases, the time increases*
  - *Written O(N)*

Is there a faster way?

- Yes!
- Binary search…

Demo: binary search

- **How much faster?**



```
** NU open street map **

Enter map filename>
nu.osm
# of nodes: 15070
# of buildings: 103

Enter building name (partial or complete), or * to list, or $ to end>
University Hall
University Hall
Address: 1897 Sheridan Road
Building ID: 33908928
Nodes:
  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)

Enter building name (partial or complete), or * to list, or $ to end>
$

** Done  **
# of calls to getID()  316
# of Nodes created: 15070
# of Nodes copied: 31453
```

Binary search

2	4	9	14	18	22	36	54	71	88	101

- **Jump to the middle**
- **Compare --- if == stop, if < go left, if > go right**
- **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 $\log_2 N$"**
  - *as N increases, the time increases much more slowly*
  - *Written O(lgN)*

Interview question

- There are 65,000 elements in a vector, in random order
- You need to perform 4 searches of this data
- What is the most efficient approach?

A) Use linear search

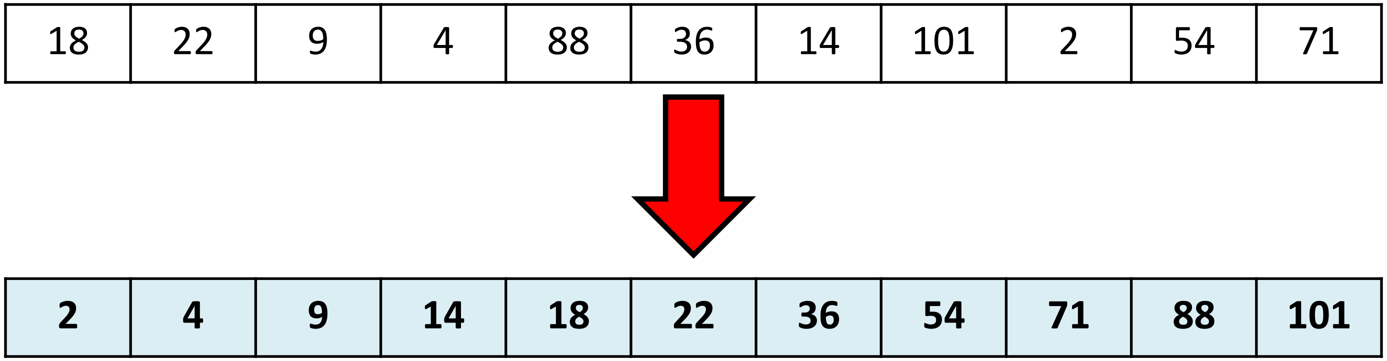
B) Use binary search

C) Sort then use linear search

D) Sort then use binary search

Binary search pre-condition: sorted order

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

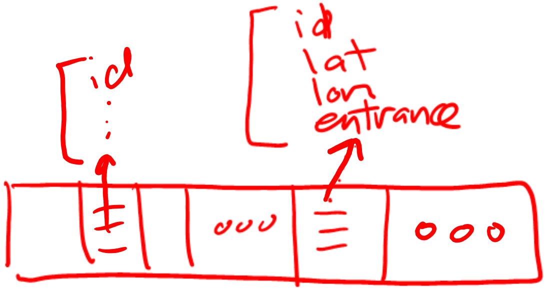


18	22	9	4	88	36	14	101	2	54	71

2	**4**	**9**	**14**	**18**	**22**	**36**	**54**	**71**	**88**	**101**

Sorting in C++

- **Built-in algorithm**
- **Uses lambda function to sort objects**

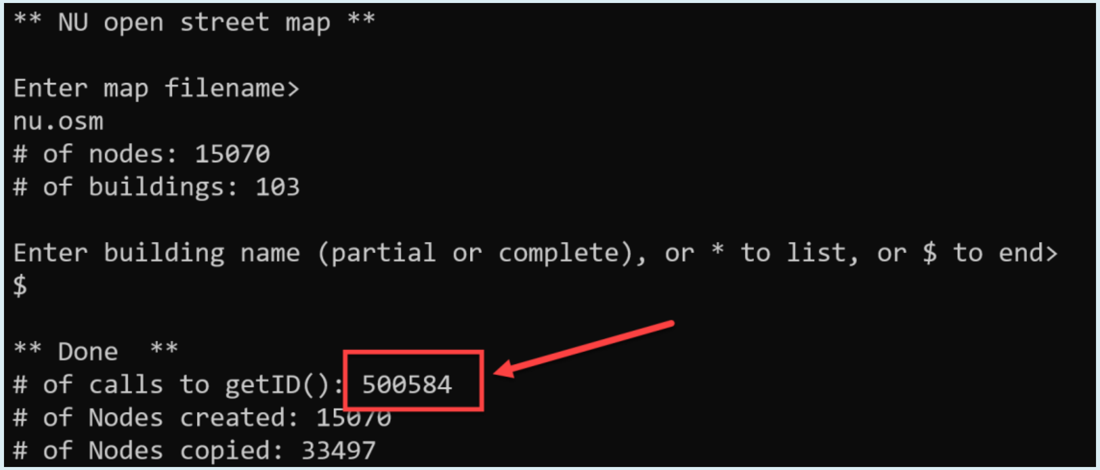


```
#include <algorithm>
```

```
std::sort(this->MapNodes.begin(), this->MapNodes.end(),

    [](const Node& node1, const Node& node2)
    {
       if (node1.getID() < node2.getID()) // keep this order
         return true;
       else // swap them
         return false;
    }

);
```

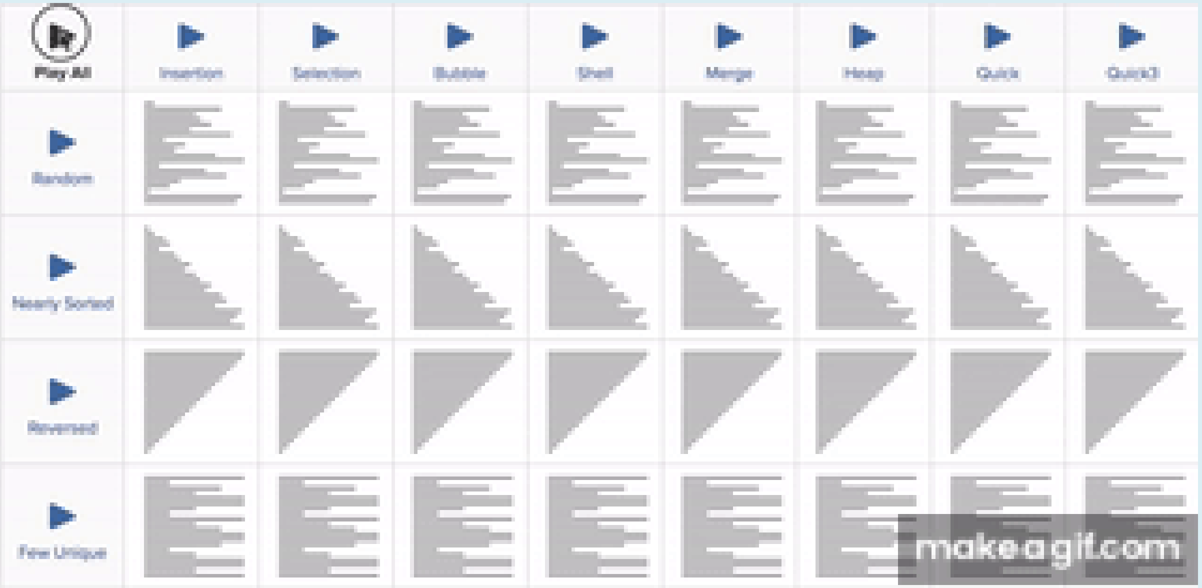
Demo: sorting

- **What's the cost of sorting the nodes?**



```
** NU open street map **

Enter map filename>
nu.osm
# of nodes: 15070
# of buildings: 103

Enter building name (partial or complete), or * to list, or $ to end>
$

** Done  **
# of calls to getID(): 500584
# of Nodes created: 15070
# of Nodes copied: 33497
```

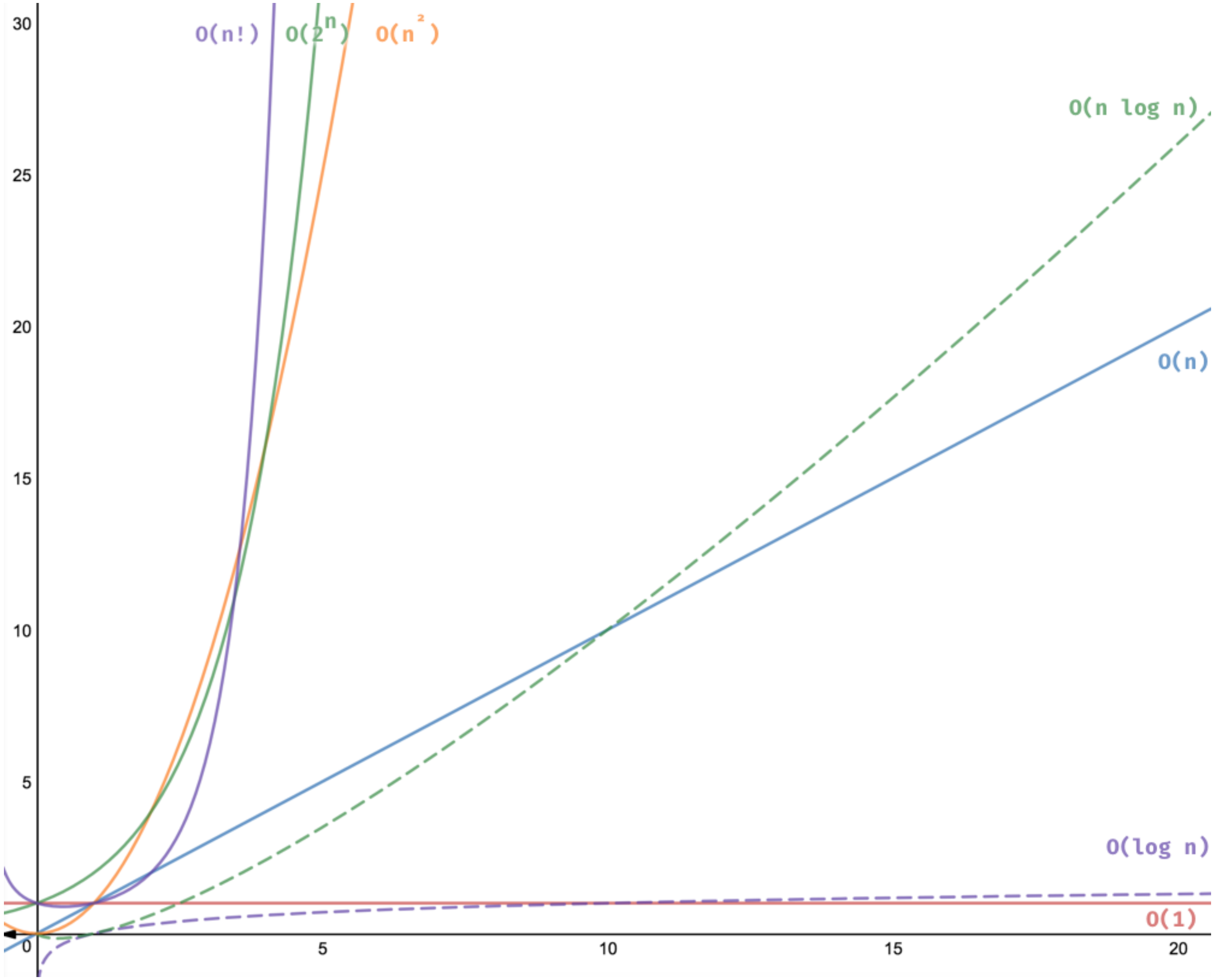
Sorting Algorithms



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

Time complexities



Time

Linear search

Binary search

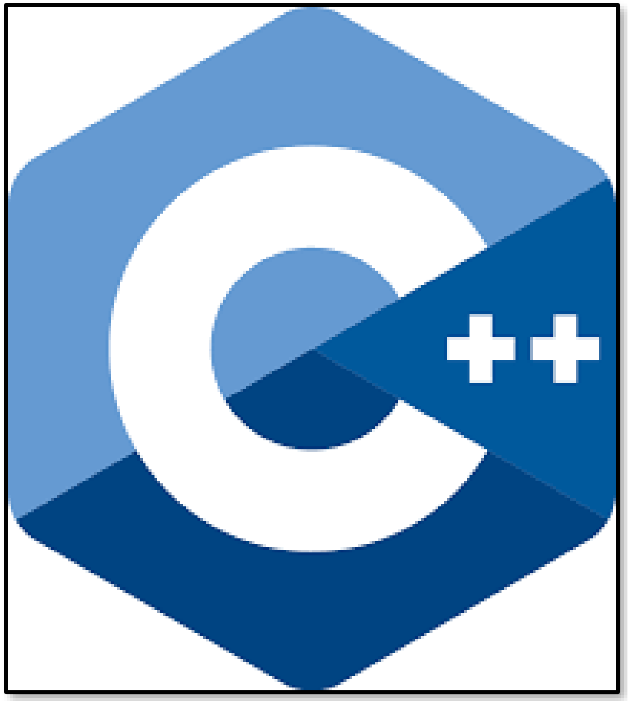
N (amount of data)

Does time complexity really matter?

Example:

- *Suppose **N = 1,000,000 (1MB)***

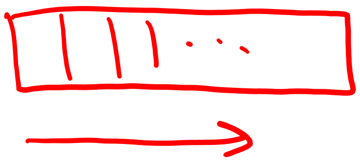
	Time Complexity	# of steps	Example Algorithm
Algorithm A	O(1)		
Algorithm B	O(lgN)		
Algorithm C	O(N)		
Algorithm D	O(NlgN)		
Algorithm E	O(N^2)		


C++

Searching

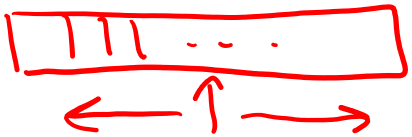
Searching

linear search



O(n) per search

sort + binary search



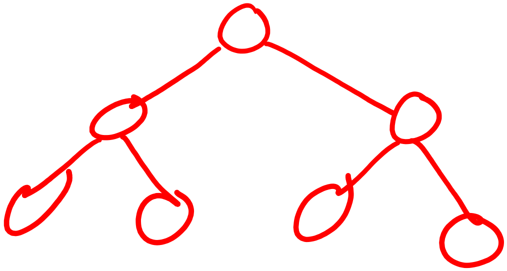
O(nlgn) sort

O(lgn) per search

- - - - - - - - - - - - - - - - - - -

O(n) insert/delete

search trees



O(nlgn) to build

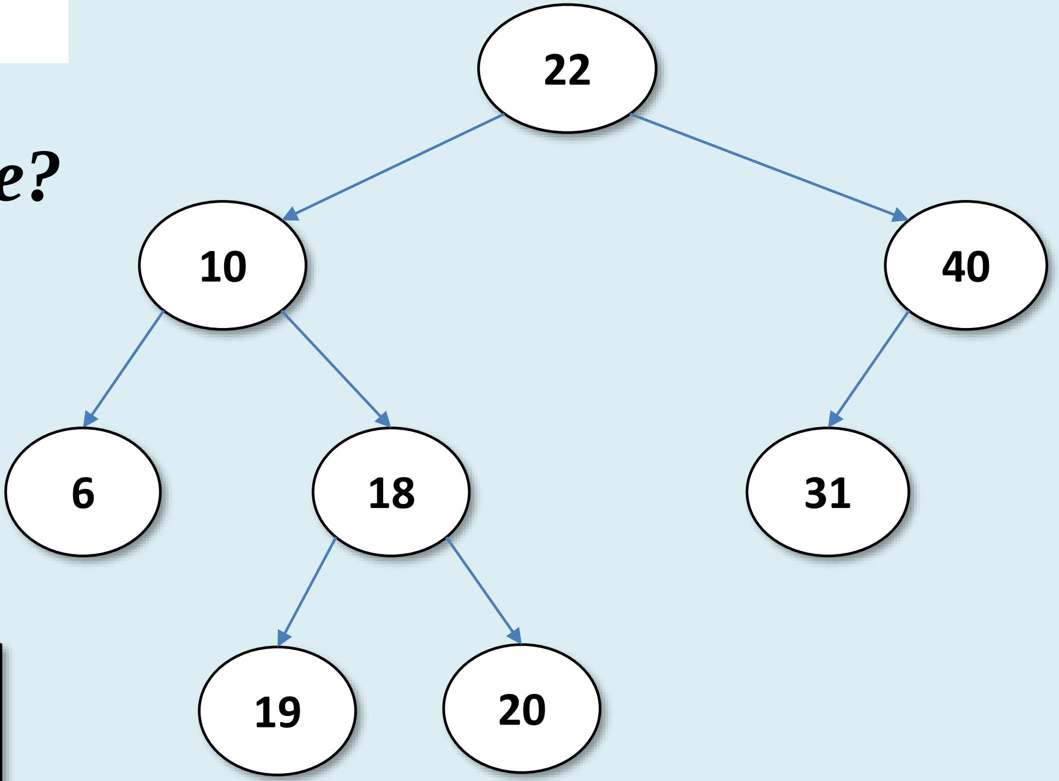
O(lgn) per search

- - - - - - - - - - - - - - - - - - -

O(lgn) insert/delete

Question

Is this a valid search tree?



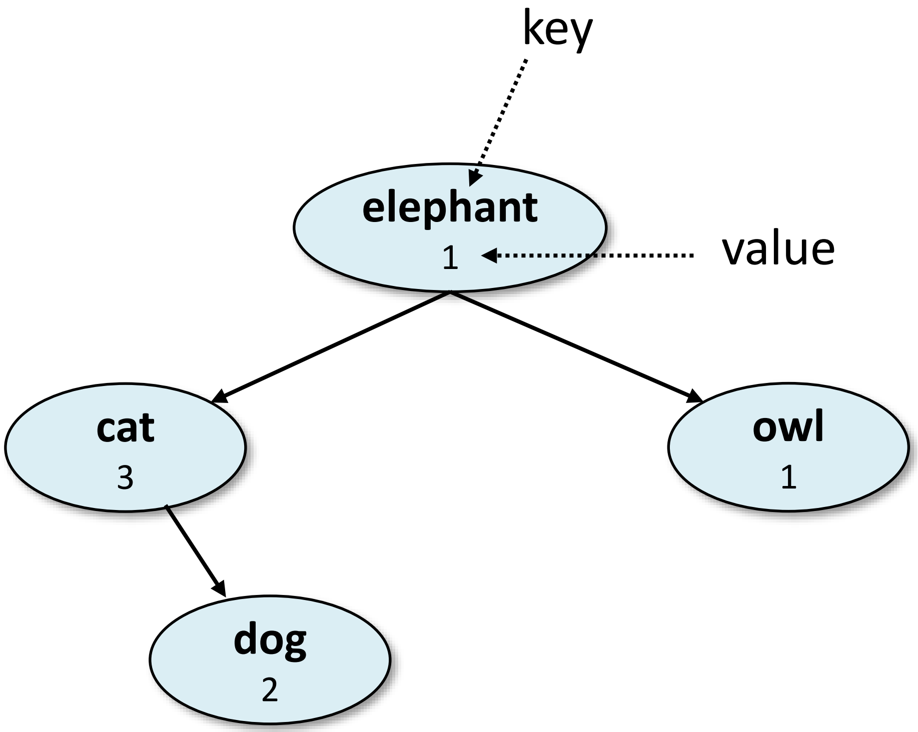
- *A) yes*
- *B) no, 18 is out of place*
- *C) no, 19 is out of place*
- *D) no, 31 is out of place*

map

- **map is C++ abstraction for a search tree**
  - *"map" key to value*

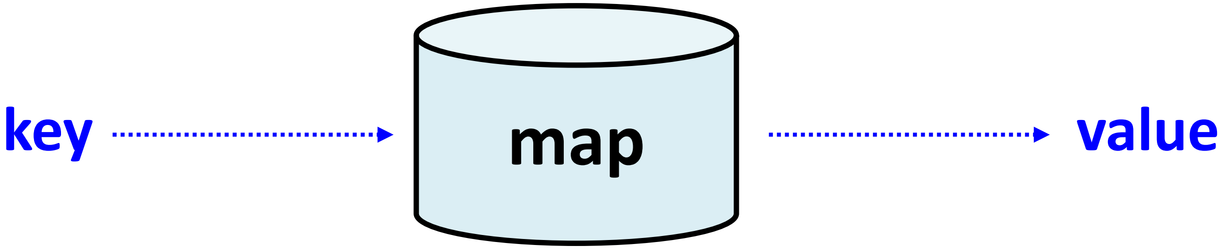
```
#include <map>
```

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

   animals["elephant" ] = 1;
   animals["cat"] = 3;
   animals["owl"] = 1;
   animals["dog"] = 2;
   .
   .
   .
```



<key, value> pairs

- Designed for fast O(lgN) lookup by key



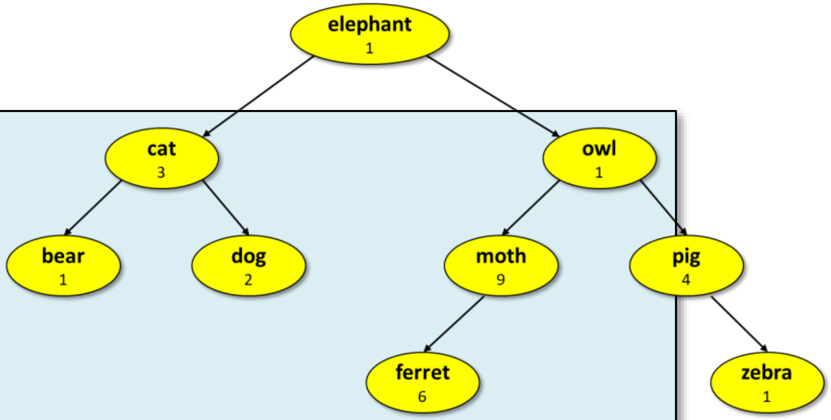
key

map

value

key value

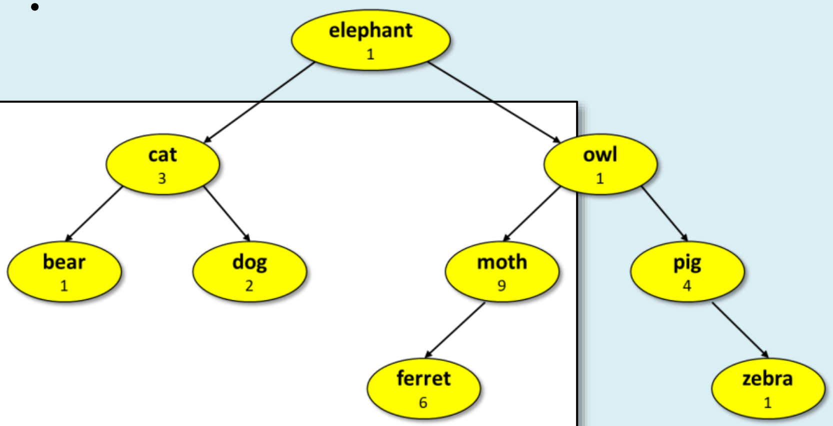
```
int main()
{
   map<string, int>  animals;
   .
   .
   .
   string type;
   cout << "Enter a type of animal> ";
   cin >> type;

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



beware of [ ]

- **[ ] performs a search**
  - *If not found, inserts key with default value*

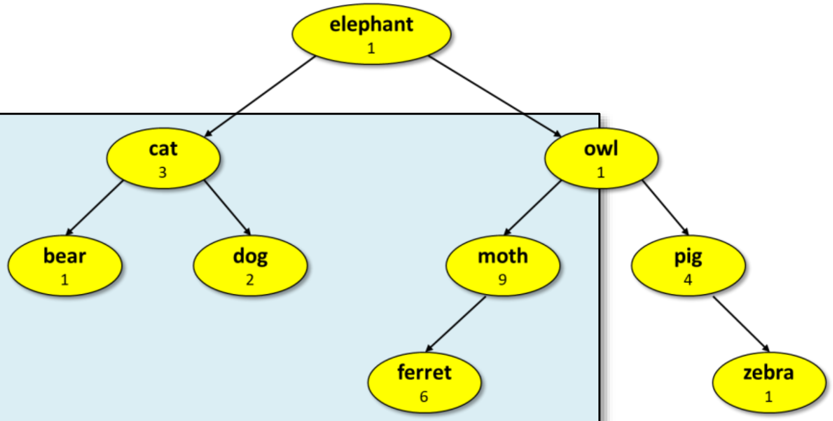
- **Example:**
  - *What if the user enters "**newt**" ?*



```
int main()
{
   map<string, int>  animals;
   .
   .
   .
   string type;
   cout << "Enter a type of animal> ";
   cin >> type;

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

   cout << animals.size() << endl;
```

prefer searching with find( )

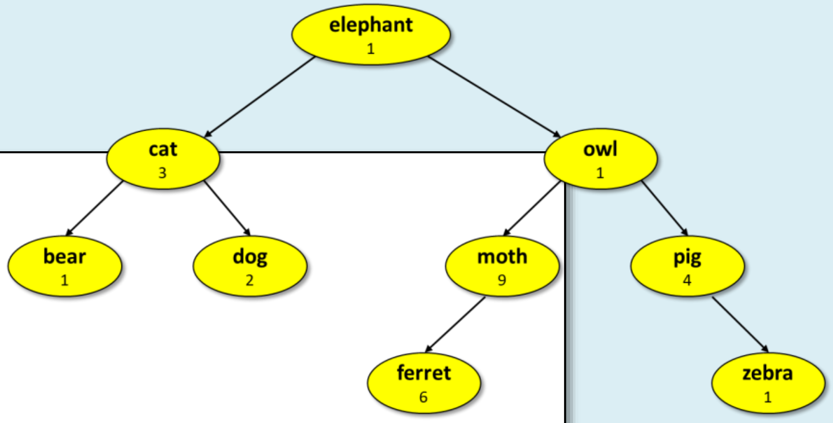
- find( ) will search and return the following:
  - *Returns an **iterator** ("pointer") to the <key, value> pair if found*
  - *Returns "end of iteration space" if not found*



```
int main()
{
   map<string, int>  animals;
   .
   .
   .
   string type;
   cout << "Enter a type of animal> ";
   cin >> type;

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

   auto iter = animals.find(type);

   if (iter == animals.end())  // not found
     cout << "I don't own this type of animal..." << endl;
   else
     cout << "I own " << iter->second << " animals of this type" << endl;
```

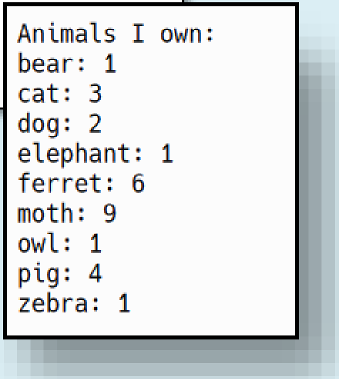
iteration

- **Interestingly, you can iterate across the tree using foreach**
  - *Traverses in key order…*



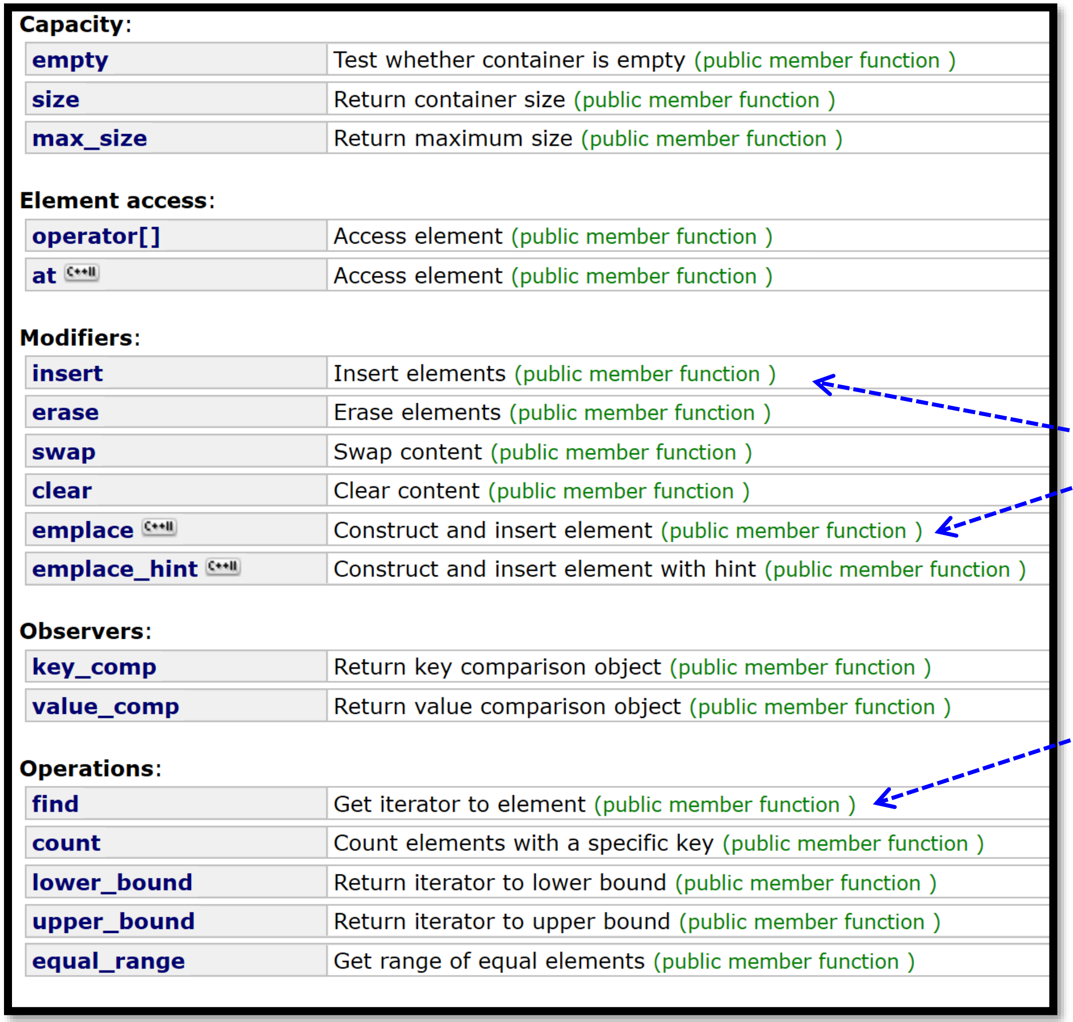
```
int main()
{
   map<string, int>  animals;
   .
   .
   .

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

   for (auto kv_pair : animals)
     cout << kv_pair.first << ": " << kv_pair.second << endl;
```

Humm, I wonder how this works? [project 08?]



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

map: API summary



Capacity:

empty	Test whether container is empty (public member function)
size	Return container size (public member function)
max_size	Return maximum size (public member function)

Element access:

operator[]	Access element (public member function)
at C++11	Access element (public member function)

Modifiers:

insert	Insert elements (public member function)
erase	Erase elements (public member function)
swap	Swap content (public member function)
clear	Clear content (public member function)
emplace C++11	Construct and insert element (public member function)
emplace_hint C++11	Construct and insert element with hint (public member function)

Observers:

key_comp	Return key comparison object (public member function)
value_comp	Return value comparison object (public member function)

Operations:

find	Get iterator to element (public member function)
count	Count elements with a specific key (public member function)
lower_bound	Return iterator to lower bound (public member function)
upper_bound	Return iterator to upper bound (public member function)
equal_range	Get range of equal elements (public member function)

lookup *(beware [ ] will insert)*

Additional ways to ***insert***

A better way to ***lookup****…*

Lookup, insert, and erase are O(lgN)

What's due?

HW 06 *is due tonight*

Project 06 *is due Friday night*

No class Thursday!