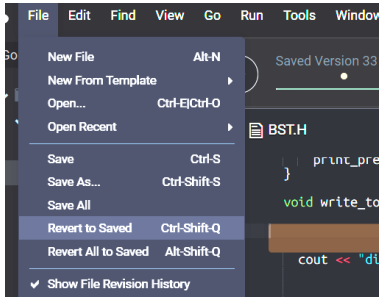
CS010C

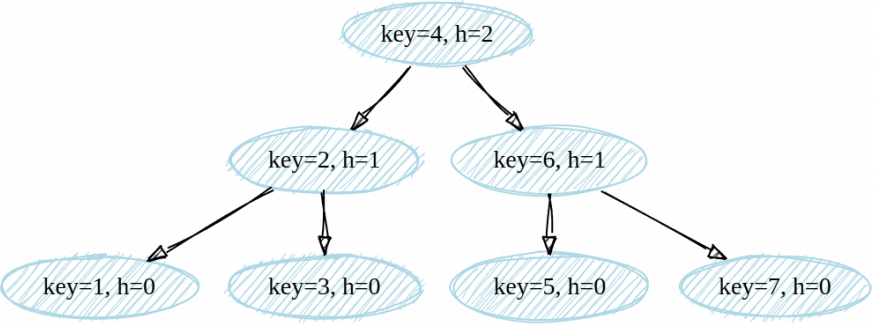
Lab7(No.7 Lab)

4th Lab Demo

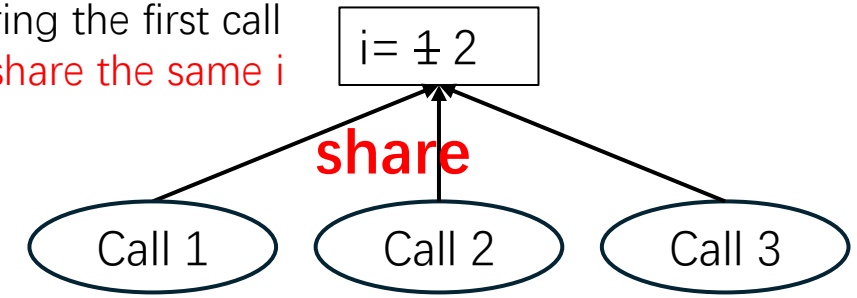


- Check file receipts (BST.H) in AWS
  - AWS menu -> File -> Show File Revision History
- Use “main_hard_code.cpp” & Use a BST provided
  - Move 1 line!
- open and run GDB
  - Just set a breakpoint
- Graphviz
  - demo one of these: pre, post, in-order (Graphviz)
  - demo one of these: depth or height (Graphviz)
- Mark on the attendance sheet: o □ G_1 G_2
  - O for log, □ for GDB, G_1 G_2 for Graphviz

```
BST t2;
t2.insert(45);
t2.insert(25);
t2.insert(65);
t2.insert(15);
t2.insert(35);
t2.insert(55);
t2.insert(75);
```



```
static int i = 1;      // executed only once, during the first call
i++;                   // all calls to the function share the same i
```



Hints

- pre, post, in-order
  - How to maintain globally incrementing number?
    1. Use “static”
       - Each visit: num++
    2. ~~Or Use global variable (not suggested)~~
- Height
  - $H = 1 + \max(leftH, rightH)$
- Depth?
  1. Use “static”
     - Visit child: depth++
     - Befor return: depth--
  2. ~~Or use 1 more parameter~~ ~~`set_depth(BinaryNode* t, int depth)`~~

Lab5 Assignment

https://learn.zybooks.com/zybook/UCRCS010CRusichSummerSessionB2025/chapter/5/section/18

Min-Heap Review

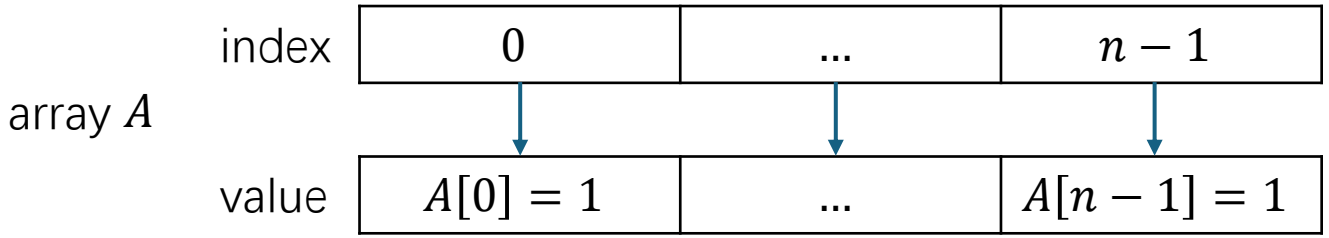
- See slides
  - CANVAS -> modules -> week4 -> Binary Heap Slides
- We have n nodes, from 0 to $n-1$
  - Node i's
    - Parent = $(i-1)/2$
    - Left Child = $2i+1$
    - Right Child = $2i+2$
  - Insert
    - Put the new node at n-th position, then trickle up
      - (iteratively compare with parent)
  - Remove_min
    - Remove the root, place last item into the hole, then trickle down
      - (iteratively compare with the smaller child - remember 2 children)
- Build heap
  - $O(n)$, rather than $O(n\log n)$

```
// Trickle up
i = numNodes // i == size
while (i > 0 && h[(i-1)/2] > item)
    h[i] = h[(i-1)/2]
    i = (i-1)/2
 h[i] = item
 numNodes++
```

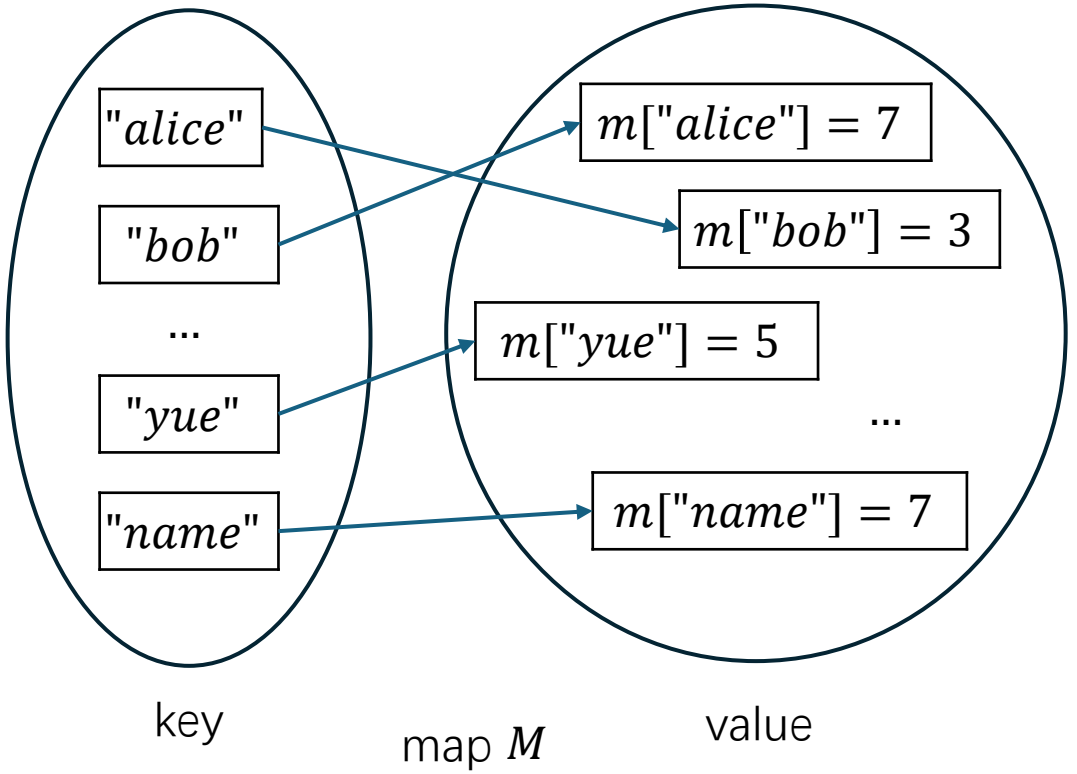
Remove highest priority item

- Makes a hole at the root
- Want to remain a complete tree, so attempt to place last item in the heap into the hole
  - If item can be placed in hole without violation of the heap property, then done
  - Otherwise, trickle down
  - **Pick the child with the highest priority - two children**

map



- Review: what is an array? consists of index-value pairs
  - an $index$, must be integer
  - you can get the $value$ at the corresponding $index$'s position
  - $index$ is unique, $value$ can be repeated
- A map: consists of key-value pairs
  - a key, can be anything
  - you can look up the $value$ by the key
  - key is unique, $value$ can be repeated
- Map's underlying implementation
  - A kind of BST
  - very complex
    - Don't focus on it
    - Just use it (std::map)



std::map example

```cpp
#include <iostream>
#include <map>
#include <string>

int main ()
{
    std::map<char,std::string> mymap;

    mymap['a'] = "an element";
    mymap['b'] = "another element";
    mymap['c'] =  mymap['b'];

    std::cout << "mymap['a'] is " << mymap['a'] << '\n';
    std::cout << "mymap['b'] is " << mymap['b'] << '\n';
    std::cout << "mymap['c'] is " << mymap['c'] << '\n';
    std::cout << "mymap now contains " << mymap.size() << " elements.\n";

    std::cout << "-----UPDATE mymap['c']-----\n";
    mymap['c'] = "something else";
    std::cout << "mymap['c'] is " << mymap['c'] << '\n';
    std::cout << "mymap now contains " << mymap.size() << " elements.\n";

    std::cout << "-----WHAT IF mymap['d'] IS NOT SET BUT IS ACCESSED-----\n";
    std::cout << "mymap['d'] is " << mymap['d'] << '\n';
    std::cout << "mymap now contains " << mymap.size() << " elements.\n";
    return 0;
}
```

set

get

size

Example:
https://github.com/SJZHZ/UCR_CS010C_25U/blob/main/demos/Lab7/map.cpp

```
❯ bash run.sh
mymap['a'] is an element
mymap['b'] is another element
mymap['c'] is another element
mymap now contains 3 elements.
-----UPDATE mymap['c']-----
mymap['c'] is something else
mymap now contains 3 elements.
-----WHAT IF mymap['d'] IS NOT SET BUT IS ACCESSED-----
mymap['d'] is
mymap now contains 4 elements.
 ~/UCR_CS010C_25U/demos/Lab7  main !2 ?3
```

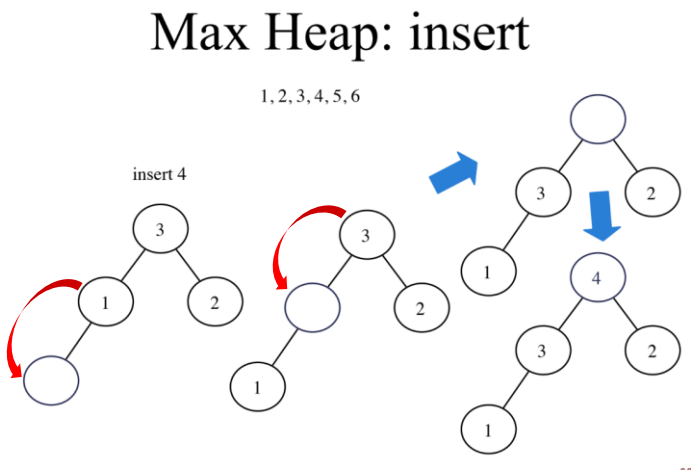
pq_zero.H

- Private
  - Member variables (completed)
    - “heap” vector stores <Item>s
    - “index” map stores <Item,int> pairs
    - “priority” map stores <Item,float> pairs
  - Basic Operations (relatively complex)
    - `void percolate_up( indx i );`
    - `void percolate_down( indx i );`
- Public member functions (straightforward)
  - Once you have completed the basic operations

```
private:
  vector<Item> heap; // The heap expands/shrinks to fit data
  typedef int indx;  // index with heap
  map<Item,indx> index;  // records each Item's place in heap
  map<Item,float> priority; // records each Item's priority
  void percolate_up( indx i );
  void percolate_down( indx i );
```

```
public:
  // These use the min-heap functions above.
  int size( ) const;
  bool empty( ) const;
  const Item& front( ) const;
  void pop( );
  void push( const Item& w, float prio );
};
```

heap & PQ



- Different element
  - Heap: only a priority value.
  - PQ: a priority value and other content
- Different operation
  - percolate_up, for example
    - In heap: we have $h[i]$
      - $h[i] = h[(i-1)/2]$
    - In PQ: we have “heap”, “index”, “priority”
      - What will change?
        - “heap”
        - “index”
        - “priority”? The relationship between item & priority remains unchanged!
      - How will they change?
  - Likewise for percolate_down

```
vector<Item> heap; // The
map<Item,indx> index;  //
map<Item,float> priority;
```