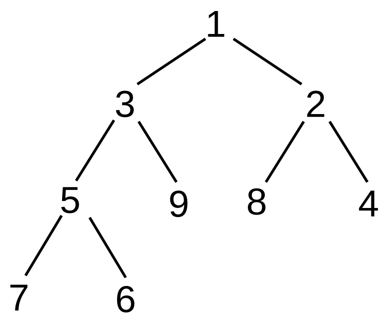
Homework #3

Due date: 4/5 lab

Heapsort

A *heap* is a nearly complete binary tree, as shown below. The tree is completely filled on all levels except possibly the lowest, which is filled from the left up to a point.



There are two kinds of heaps: min-heaps and max-heaps.

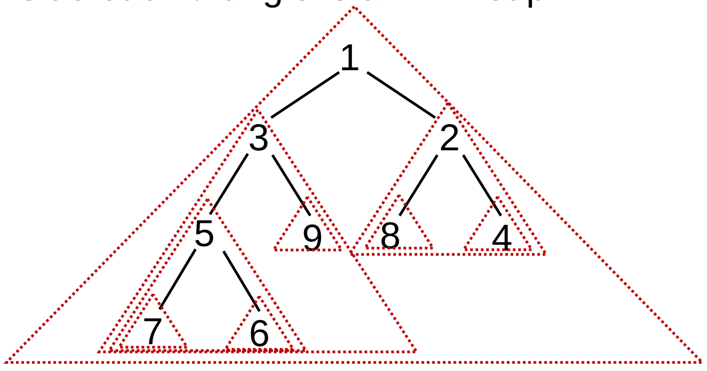
A *min-heap* gives the minimum element the highest priority, i.e. it satisfies the min-heap property:

element in the root $\leq$ *elements in both subtrees*

A *max-heap* gives the maximum element the highest priority, i.e. it satisfies the max-heap property:

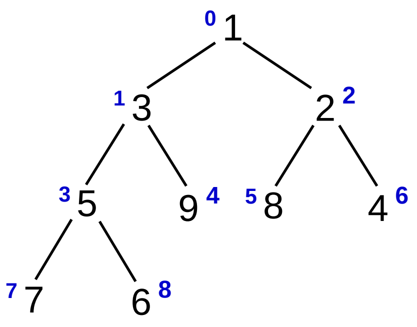
element in the root $\geq$ *elements in both subtrees*

For example, the preceding heap is a min-heap. Notice that every subtree of a min-heap is also a min-heap. In the figure below, the tree inside each triangle is a min-heap.



Array implementation of heap

A *heap* may be represented by an array.
For example, the array h contains the preceding min-heap.



array h	1	3	2	5	9	8	4	7	6
Index	0	1	2	3	4	5	6	7	8

Application of heap

In this homework, you are asked to implement an application of the heap data structure, namely, the heapsort algorithm.

Although this homework is lengthy, it is in fact an ***easy*** exercise, as the code of most functions are given below. The important thing for you to do is to digest the concepts introduced in this homework. In particular, be sure you understand the following terms: heap, binary tree, subtree, root, parent, left child, and right child.

To implement heapsort, you have to write the following 5 functions.

Left child and right child

1 lchild(k) $= 2k + 1 =$ the index of the left child of node k
2 rchild(k) $= 2k + 2 =$ the index of the right child of node k

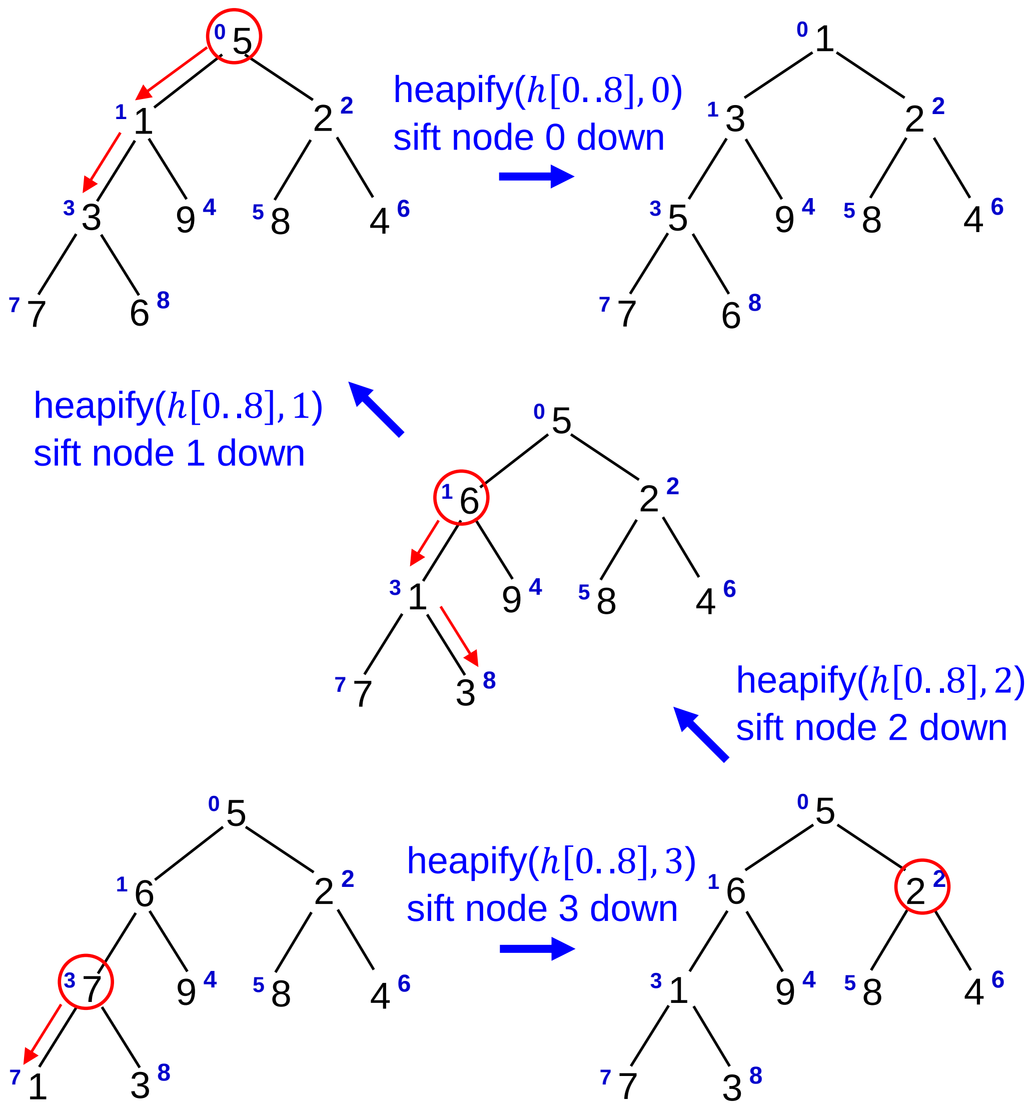
Maintaining the heap property

3 heapify($h[0..n-1], k$) // $n =$ heap size, $0 \le k \le n-1$

Precondition – The subtrees rooted at lchild(k) and rchild(k) are heaps. But, the root k itself may not satisfy the heap property.
Postcondition – The binary tree rooted at k is a heap.

Q: How does heapify work?

A: Sift node k down to its new position so that the binary tree rooted at node k becomes a heap, e.g.



Building a heap

4 buildheap($h[0..n-1]$) // turn array $h[0..n-1]$ into a heap

```
for $k = n/2 - 1$ downto $0$ do
    heapify($h[0..n-1], k$)
```

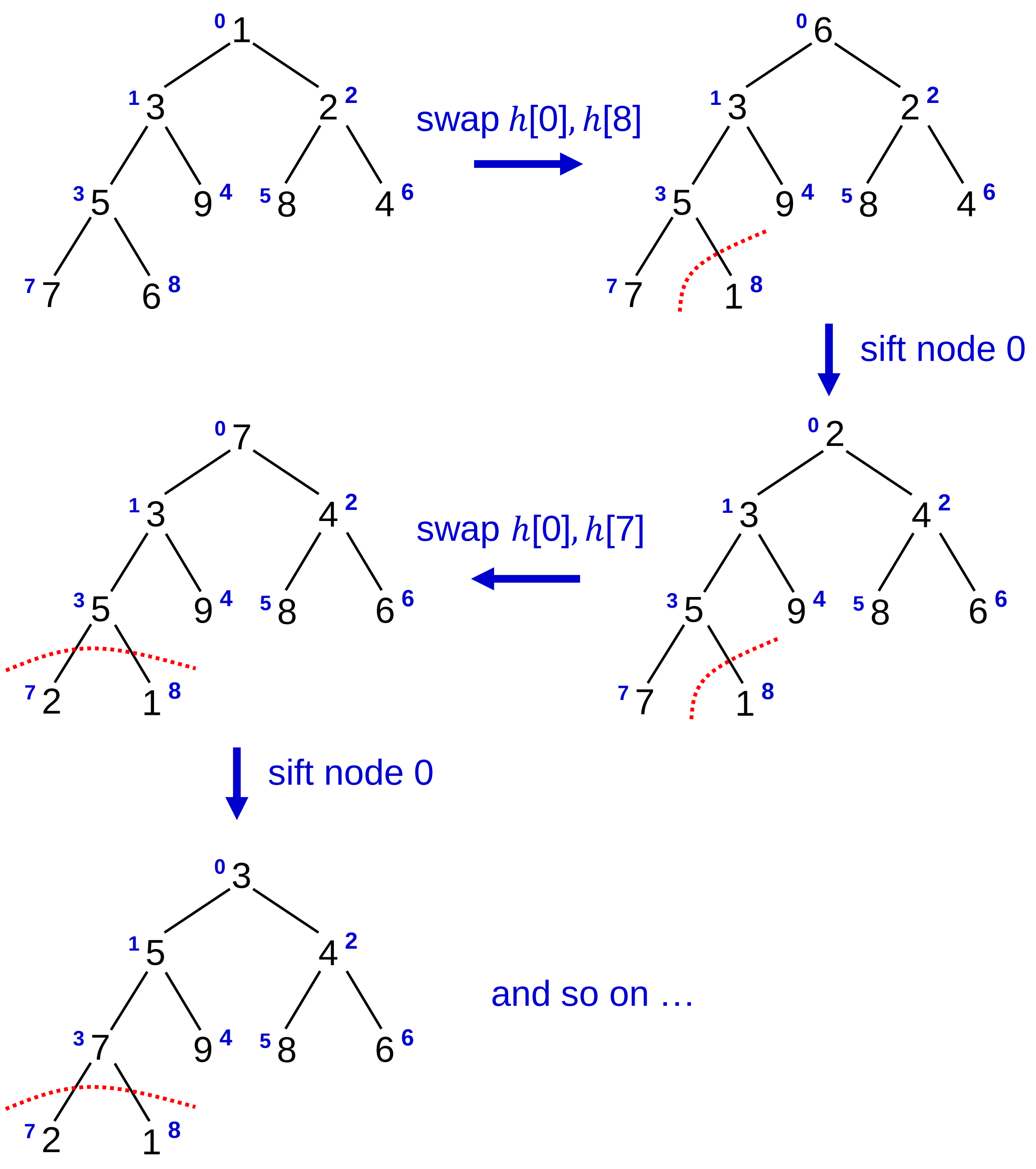
Example

The process above turns the array $\langle 5, 6, 2, 7, 9, 8, 4, 1, 3 \rangle$ to a heap $\langle 1, 3, 2, 5, 9, 8, 4, 7, 6 \rangle$ step-by-step.

The heapsort algorithm

```
6   heapsort($h[0..n-1]$)          // sort the array $h[0..n-1]$

    1   buildheap($h[0..n-1]$)
    2   for $k = n-1$ downto $1$ do
           swap $h[0]$ and $h[k]$
           heapify($h[0..k-1], 0$)
```



Observation

min-heap ⇒ non-increasing order
max-heap ⇒ non-decreasing order

Requirements

1 *lchild* and *rchild* shall be defined by lambda expressions within the function *heapify*.
heapify, *buildheap*, and *heapsort* shall be function templates, as specified below.

2 Write the following STL-style function template

```
template<typename T,typename Compare>
void heapsort(T* begin,T* end,Compare comp);
```

to sort the elements in the sequence **[begin,end)** according to the comparison function **comp**.

If **comp** is $<$, the elements shall be sorted into non-decreasing order by employing a max-heap.

If **comp** is $>$, the elements shall be sorted into non-increasing order by employing a min-heap.

Q: Can we use $\leq$ or $\geq$?
A: No. The behavior is undefined in STL. To be consistent with STL, you shall use $<$ and $>$.

Q: Let $x_1, x_2, \cdots, x_n$ be the n elements in the sorted sequence. As described above, if **comp** is $<$, then $x_1 \leq x_2 \leq \cdots \leq x_n$. How can we use $<$ to obtain a result expressed by $\leq$?
A: Well, this is easy, since
$x_1 \leq x_2 \leq \cdots \leq x_n$ (expressed by $\leq$)
is equivalent to
$x_n \nless \cdots \nless x_2 \nless x_1$ (expressed by $<$ and logical negation)

3 [Option]
You may also define an overloaded function template

```
template<typename T,typename Compare>
void heapsort(T* h,int n,Compare comp);
```

to sort the array **h**[**0**..**n-1**] according to **comp**.
If it is defined, it shall be called by the STL-style **heapsort** mentioned above.

4 You shall also write the following two function templates.

```
// Heapify node k of h[0..n-1] according to comp
template<typename T,typename Compare>
void heapify(T* h,int n,int k,Compare comp);
```

```
// Turn h[0..n-1] to a heap according to comp
template<typename T,typename Compare>
void buildheap(T* h,int n,Compare comp);
```

5 Q: How to express <?
A: There are three methods.
Method 1: Use ordinary function, say

```
bool less(int x,int y) { return x<y; }
```

Method 2: Use lambda expression, say

```
[](int x,int y){ return x<y; }
```

Method 3: Use STL comparison functor, say

```
less<int>()
```

Comments

1 A comparison functor is a function object that supports <, <=, >,>=, ==, or !=.

2 The STL class template **less** is defined as

```
template<typename T>
struct less
{
   bool operator()(const T& x,const T& y) const
   {
      return x<y;
   }
};
```

3 Other class templates for comparison functors

```
greater        >
less_equal     <=    equal_to       =
greater_equal  >=    not_equal_to   !=
```

6 For testing purpose, you are asked to

1 express $>$ by lambda expression, and
2 express $<$ by STL comparison functor.

For the latter case, in order to sort sequences of C-style strings by $<$, you shall add the following explicit specialization to the **std** namespace.

```
namespace std {
   template<> struct less<const char*>;
};
```

Comments

1 An explicit specialization must be defined in the namespace of which the template it specializes is a member.
Thus, the specialization of **less** shall be defined in the **std** namespace.

Note: GNU C++ meets this requirement, but VC++ doesn't. In other words, if the specialization is mistakenly defined in the global namespace, it will still compile in VC++.

2 A namespace may be enlarged anywhere in a program. In particular, the user may enlarge the namespace **std**.

Sample test

The sample test given in file hw3test.cpp contains an incomplete function **main**, as shown below.
You shall fill in the blanks to make it yield the sample output.

```
#include <iostream>
#include <cstring>
#include <algorithm>
#include <functional>
using namespace std;
```

```
int main()
{
    int a[15]={4,5,2,5,4,3,5,4,1,3,3,5,5,4,2};
    heapsort(a,a+15,_____);     // fill in a comparison functor
    _____     // use STL function for_each to print out the array a

    char b[]="SnoopyLikesC++!Crazy!";
    heapsort(b,b+sizeof(b)-1,_____);     // fill in a lambda expression
    cout << b << endl;

    // fill in a polymorphic function wrapper, a comparison functor, and a lambda
    // expression, in that order
    _____ comp[2] = {_____,_____};
    for (int i=0;i<2;i++) {
        const char* c[11]={"Knicks","Lakers","76ers","Hawks",
                    "Bulls","Mavericks","Heat","Wizards",
                    "Bucks","Pacers","Magic"};
        heapsort(c,c+11,comp[i]);
        _____     // use STL function for_each to print out the array c
    }
}
```

Sample output

```
1 2 2 3 3 3 4 4 4 4 5 5 5 5 5
zyysrpoonkieaSLCC++!!
76ers Bucks Bulls Hawks Heat Knicks Lakers Magic Mavericks Pacers Wizards
Wizards Pacers Mavericks Magic Lakers Knicks Heat Hawks Bulls Bucks 76ers
```

Final remarks

1 The STL contains a function template **sort** in **<algorithm>**. You may test it by replacing **heapsort** by **sort** in the sample test. The resulting program shall still work.

2 On the other hand, if you also replace **less** by **less_equal**, the program will cause a run time error in VC++, but still work in GNU C++.
In this regard, VC++ meets the requirement, but GNU C++ fails.