

Week 04

Priority Queues, binary heaps,
Symbol Tables, ordered, unordered,
BST's

PQ API

```
public class MaxPQ<Key> extends Comparable<Key> {
    MaxPQ() create an empty priority queue
    MaxPQ(Key[] a) create a priority queue with given keys
    void insert(Key v) insert a key into the priority queue
    Key delMax() return and remove a largest key
    boolean isEmpty() is the priority queue empty?
    Key max() return a largest key
    int size() number of entries in the priority queue
}
```

Implementation

implementation	insert	del max	max
unordered array	1	N	N
ordered array	N	1	1
goal	$\log N$	$\log N$	$\log N$

Binary Heap Implementation

Heap-ordered binary tree.

- Keys in nodes.
- Parent's key no smaller than children's keys.

Array representation.

- Indices start at 1.
- Take nodes in **level** order.
- No explicit links needed!

```
private void sink(int k)
{
    while (2*k <= N)
    {
        int j = 2*k;
        if (j < N && less(j, j+1)) j++;
        if (!less(k, j)) break;
        exch(k, j);
        k = j;
    }
}
```

*children of node at k are 2*k and 2*k+1*

```
private void swim(int k)
{
    while (k > 1 && less(k/2, k))
    {
        exch(k, k/2);
        k = k/2;
    }
}
```

parent of node at k is at k/2

Priority queues: quiz 1

How many compares (in the worst case) to **insert** in a d -way heap?

- A. $\sim \log_2 N$
- B. $\sim \log_d N$
- C. $\sim d \log_2 N$
- D. $\sim d \log_d N$
- E. *I don't know.*

Implementation cost summary

implementation	insert	del max	max
unordered array	1	N	N
ordered array	N	1	1
binary heap	$\log N$	$\log N$	1
d -ary heap	$\log_d N$	$d \log_d N$	1
Fibonacci	1	$\log N^{\dagger}$	1
Brodal queue	1	$\log N$	1
impossible	1	1	1

← sweet spot: $d = 4$

← why impossible?

\dagger amortized

order-of-growth of running time for priority queue with N items

Priority queues: quiz 3

What is the name of this sorting algorithm?

```
public void sort(String[] a)
{
    int N = a.length;
    MaxPQ<String> pq = new MaxPQ<String>();
    for (int i = 0; i < N; i++)
        pq.insert(a[i]);
    for (int i = N-1; i >= 0; i--)
        a[i] = pq.delMax();
}
```

- A. Insertion sort.
- B. Mergesort.
- C. Quicksort.
- D. *None of the above.*
- E. *I don't know.*

Priority queues: quiz 4

What are its properties?

```
public void sort(String[] a)
{
    int N = a.length;
    MaxPQ<String> pq = new MaxPQ<String>();
    for (int i = 0; i < N; i++)
        pq.insert(a[i]);
    for (int i = N-1; i >= 0; i--)
        a[i] = pq.delMax();
}
```

- A. $N \log N$ compares in the worst case.
- B. In-place.
- C. Stable.
- D. *All of the above.*
- E. *I don't know.*

Sorting algorithms: summary

	inplace?	stable?	best	average	worst	remarks
selection	✓		$\frac{1}{2} N^2$	$\frac{1}{2} N^2$	$\frac{1}{2} N^2$	N exchanges
insertion	✓	✓	N	$\frac{1}{4} N^2$	$\frac{1}{2} N^2$	use for small N or partially ordered
shell	✓		$N \log_2 N$?	$c \cdot N^{3/2}$	tight code; subquadratic
merge		✓	$\frac{1}{2} N \lg N$	$N \lg N$	$N \lg N$	$N \log N$ guarantee; stable
timsort		✓	N	$N \lg N$	$N \lg N$	improves mergesort when preexisting order
quick	✓		$N \lg N$	$2 N \ln N$	$\frac{1}{2} N^2$	$N \log N$ probabilistic guarantee; fastest in practice
3-way quick	✓		N	$2 N \ln N$	$\frac{1}{2} N^2$	improves quicksort when duplicate keys
heap	✓		N	$2 N \lg N$	$2 N \lg N$	$N \log N$ guarantee; in-place
?	✓	✓	N	$N \lg N$	$N \lg N$	holy sorting grail

Basic symbol table API

Associative array abstraction. Associate one value with each key.

```

public class ST<Key, Value>
{
    ST()                                create an empty symbol table
    void put(Key key, Value val)         put key-value pair into the table ← a[key] = val;
    Value get(Key key)                   value paired with key ← a[key]
    boolean contains(Key key)            is there a value paired with key?
    Iterable<Key> keys()                  all the keys in the table
    void delete(Key key)                 remove key (and its value) from table
    boolean isEmpty()                    is the table empty?
    int size()                           number of key-value pairs in the table
}

```

Ordered symbol table API

```

public class ST<Key extends Comparable<Key>, Value>
{
    ...
    Key min()                          smallest key
    Key max()                          largest key
    Key floor(Key key)                  largest key less than or equal to key
    Key ceiling(Key key)                smallest key greater than or equal to key
    int rank(Key key)                   number of keys less than key
    Key select(int k)                   key of rank k
    ...
}

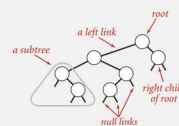
```

Binary search trees

Definition. A BST is a binary tree in symmetric order.

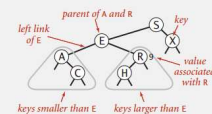
A binary tree is either:

- Empty.
- Two disjoint binary trees (left and right).



Symmetric order. Each node has a key, and every node's key is:

- Larger than all keys in its left subtree.
- Smaller than all keys in its right subtree.



Binary search trees: quiz 1

Given N distinct keys, what is the name of this sorting algorithm?

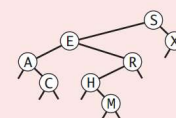
1. Shuffle the keys.
2. Insert the keys into a BST, one at a time.
3. Do an **inorder traversal** of the BST.

- A. Insertion sort.
 B. Mergesort.
 C. Quicksort.
 D. None of the above.
 E. I don't know.

Binary search trees: quiz 2

In what order does the `traverse(root)` code print out the keys in the BST?

```
private void traverse(Node x)
{
    if (x == null) return;
    traverse(x.left);
    StdOut.println(x.key);
    traverse(x.right);
}
```



- A. A C E H M R S X
 B. A C E R H M X S
 C. S E A C R H M X
 D. C A M H R E X S
 E. I don't know.

BST: ordered symbol table operations summary

	sequential search	binary search	BST
search	N	$\log N$	h
insert	N	N	h
min / max	N	1	h
floor / ceiling	N	$\log N$	h
rank	N	$\log N$	h
select	N	1	h
ordered iteration	$N \log N$	N	N

h = height of BST
 (proportional to $\log N$
 if keys inserted in random order)

order of growth of running time of ordered symbol table operations

ST implementations: summary

implementation	guarantee		average case		ordered ops?	key interface
	search	insert	search hit	insert		
sequential search (unordered list)	N	N	N	N		<code>equals()</code>
binary search (ordered array)	$\log N$	N	$\log N$	N	✓	<code>compareTo()</code>
BST	N	N	$\log N$	$\log N$	✓	<code>compareTo()</code>
red-black BST	$\log N$	$\log N$	$\log N$	$\log N$	✓	<code>compareTo()</code>

Next lecture. Guarantee logarithmic performance for all operations.