Text

```java
public class InsertionSort
{
    public static void sort(Comparable[] a) {
        int N = a.length;

        for (int i = 1; i < N; i++)
        {
            // Insert a[i] among a[i-1], a[i-2], a[i-3]... ..
            for (int j = i; j > 0 && less(a[j], a[j-1]); j--)
                exch(a, j, j-1);
        }
    }

private static void exch(Comparable[] a, int i, int j) {
    Comparable t = a[i]; a[i] = a[j]; a[j] = t;
}
```

```java
public class SelectionSort
{
    public static void sort(Comparable[] a) {
        int N = a.length;

        for (int i = 0; i < N; i++)
        {
            // Exchange a[i] with smallest entry in a[i+1...N).
            int min = i; // index of smallest entry.
            for (int j = i+1; j < N; j++)
                if (less(a[j], a[min])) min = j;
            exch(a, i, min);
        }
    }
```

initial: 21, 16, 3, 7, 23, 12
16,21,3,7,23,12
3,16,21,7,23,12
3,7,16,21,23,12
3,7,16,21,23,12
3,7,12,16,21,23

```java
public class Course  {
    private String name;
    private String prefix;
    private String prof;
    private int number;

    public Course(String p, int n){
        prefix = p.toUpperCase();
        number = n;
    }

    public String toString(){
        return prefix + " " + number;
    }
}
//attributes are private so need to use getters
@Override
public int compareTo(Course other){
    return (prefix+number).compareTo(other.getPrefix() + other.getNumber());
}
```

Show the insertion of the elements below into an symbol table structured as an ordered list.

Key	Value	List Contents
K	1	[K|1]
E	2	[E|2][K|1]
Y	3	[E|2][K|1][Y|3]
S	4	[E|2][K|1][S|4][Y|3]
Y	5	[E|2][K|1][S|4][Y|5]
M	6	[E|2][K|1][M|6][S|4][Y|5]
B	7	[B|7][E|2][K|1][M|6][S|4][Y|5]
O	8	[B|7][E|2][K|1][M|6][O|8][S|4][Y|5]
L	9	[B|7][E|2][K|1][L|9][M|6][O|8][S|4][Y|5]

Using the symbol table you created above, execute the Ordered Symbol Table

Text

Hashtables allow no null keys. Hashmaps allow one null key.
When dealing with systems having low RAM: Selection (less array writes)

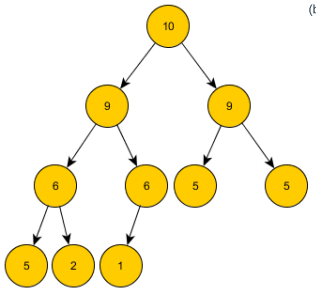
a protein structural prediction
(n-2)*(n-3) --> n^2-5n+6 --> O(n^2)

difference between an array and a symbol table?
A symbol table can use (nullable)
course number as a key and then store them into either an array or symbol table
A symbol table will probably be better. Printing a symbol table will typically only need to loop over the key/value pairs inserted.

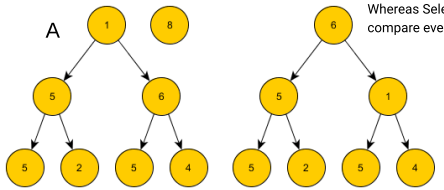
```java
    public static boolean isSorted(Comparable[] a) {
        for (int i = 1; i < a.length; i++)
            if (less(a[i], a[i-1]))
                return false;

        return true;
    }
Integer[] b = {21, 16, 3, 7, 23, 12};
```

Trace:
i=0: 16,21,3,7,23,12
i=1: 3,16,21,7,23,12
i=2: 3,7,16,21,23,12
i=3: 3,7,16,21,23,12
i=4: 3,7,12,16,21,23

5, 9, 1, 5, 6, 2, 5, 6, 9, 10. Draw a binary heap



(a) Show the steps involved in removing the maximum and then preserving the heap properties.
(b) Show the steps involved in adding a new node 9 and then preserving the heap properties.

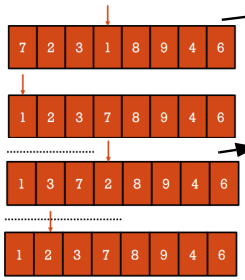


Why is it that Insertion Sort, on average, can perform better than Selection Sort
if an array is randomly sorted there is a good chance it is partially sorted, so on average, Insertion Sort has to compare fewer values. Whereas Selection Sort always has to compare every value.

algorithm	best	average	worst
selection sort	N^2	N^2	N^2
insertion sort	N	N^2	N^2
Shellsort (3x+1)	$N \log N$	?	$N^{3/2}$

Shell Sort:

Perform an insertion sort starting at the ends of the array, and gradually shrink the internal until it is 1 (i.e., converges to normal insertion sort). ▪ Moves far away things first

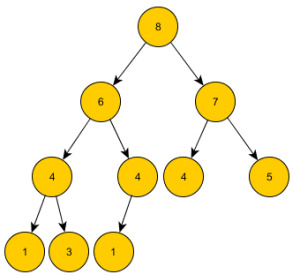


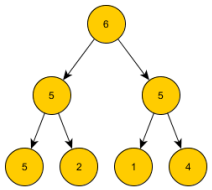
Selection Sort
1. Find smallest element in data, starting at position i
2. Swap it with the position i
3. Repeat for all elements in list
Compares: (n^2)/2
Exchanges: n

Insertion Sort
(Assume there is a region of sorted elements at the front.)
1. Pick some element at position j.
2. Insert the element into a sorted position in the sorted region.
3. Repeat for all elements in the list.
Compares: rand/reverse input: ~(n^2)/2, sorted: n-1
Exchanges: rand: ~(n^2)/4, reverse: ~(n^2)/2
sorted: 0

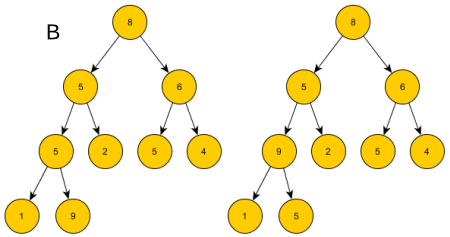
```java
public class Mergesort { O(nlogn)

    public static void sort(Comparable[] a) {
        Comparable[] aux = new Comparable[a.length];
        sort(a, aux, lo: 0, hi: a.length-1);

        assert isSorted(a);
    }

    private static void sort(Comparable[] a, Comparable[]
            aux, int lo, int hi) {
        if (hi <= lo) return;
        int mid = lo + (hi - lo) / 2;

        sort(a, aux, lo, mid);
        sort(a, aux, lo: mid + 1, hi);
        merge(a, aux, lo, mid, hi);
    }
    public static void merge(Comparable[] a, Comparable[]
            aux, int lo, int mid, int hi)
    {
        assert isSorted(a, lo, mid);
        assert isSorted(a, lo: mid+1, hi);

        int i = lo, j = mid+1;

        for(int k = lo; k <= hi; k++)
            aux[k]=a[k];

        // merge back to a[]
        for (int k = lo; k <= hi; k++) {
            if      (i > mid)              a[k] = aux[j++];
            else if (j > hi)               a[k] = aux[i++];
            else if (less(aux[j], aux[i])) a[k] = aux[j++];
            else                           a[k] = aux[i++];
        }

        assert isSorted(a, lo, hi);
    }
```

Consider the following priority queue array: [0, 8, 6, 7, 4, 4, 4, 5, 1, 3, 1]. Draw the exact binary heap for this data

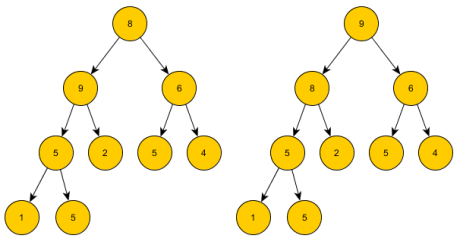




If you need to sort a large dataset on a system with limited memory would it be safe to use mergesort?
No - mergesort requires an auxiliary array to do the merging step.



initial: 2, 13, 16, 3, 7, 23, 12, 25
1 (initial): 2, 13, 16, 3, 7, 23, 12, 25
2 (down): |2, 13, 16, 3| 7, 23, 12, 25|
3 (down): |2,13|16,3|7,23|12,25|
4 (mid): |2|13|16|3|7|23|12|25|
5(up): |2,13|3,16|7,23|12,25|
6(up): |2,3,13,16|7,12,23,25|
7(up):2,3,7,12,13,16,23,25




```java
    //helper
    private static boolean less(Comparable v, Comparable w) {
        return v.compareTo(w) < 0;
    }

    private static void show(Comparable[] a) {
        for(int i = 0; i < a.length; i++)
            System.out.print(a[i] + " ");
        System.out.println();
    }

    public static boolean isSorted(Comparable[] a) {...}

    private static boolean isSorted(Comparable[] a, int lo, int hi) {
        for (int i = lo + 1; i <= hi; i++)
            if (less(a[i], a[i-1])) return false;
        return true;
    }
```

```java
public class ShellSort
{
    public static void sort(Comparable[] a) {
        int N = a.length;
        int h = 1;

        while (h < N/3) h = 3*h + 1; // 1, 4, 13, 40, 121, 364, 1093, ...

        while (h >= 1) {
            // h-sort the array.
            for (int i = h; i < N; i++) {
                // Insert a[i] among a[i-h], a[i-2*h], a[i-3*h]... .
                for (int j = i; j >= h && less(a[j], a[j-h]); j -= h)
                exch(a, j, j-h);
            }
            h = h/3;
        }
    }

    //methods below this point come from SortPlatform
    private static boolean less(Comparable v, Comparable w) {
        return v.compareTo(w) < 0;
    }

    //helper
    private static void exch(Comparable[] a, int i, int j) {...}

    private static void show(Comparable[] a) {...}

    public static boolean isSorted(Comparable[] a) {...}
```

Text