Selection sort

Function **selectionsort**. Given an array A of n numbers, indexed 0 to $n-1$

```
• for $i$ from 0 to $n-2$
•     $k = i$, this is the index of the smallest value so far
•     for $j$ from $i+1$ to $n-1$
•         if $A[j] < A[k]$ then $k = j$
•     swap $A[i]$ and $A[k]$
```

Insertion sort

Function **insertionsort**. Given an array A of n numbers, indexed 0 to $n-1$

```
• for $i$ from 1 to $n-1$
• Set $s = A[i]$
• insert $s$ into list $A[0]$ to $A[i-1]$:
•     for $j$ from $i-1$ to 0 by -1 while $A[j] > s$
•         set $A[j+1] = A[j]$
•     set $A[j+1] = s$
```

n^2 Bubblesort

Function **bubblesort**. Given an array A of n numbers, indexed 0 to $n-1$

```
• for $i$ from 0 to $n-2$
•     for $j$ from $0$ to $n-2$
•         if $A[j] > A[j+1]$ then swap them
```

$n^2/2$ Bubblesort

Function **bubblesort**. Given an array A of n numbers, indexed 0 to $n-1$

```
• for $i$ from 0 to $n-2$
•     for $j$ from $0$ to $n-i-2$
•         if $A[j] > A[j+1]$ swap them
```

Early Exit Bubblesort

Function **bubblesort**. Given an array A of n numbers, indexed 0 to $n-1$

```
• for $i$ from 0 to $n-2$
•     for $j$ from $0$ to $n-i-1$
•         if $A[j] > A[j+1]$ swap them
•     if we have done no swaps in this loop, then stop
```

Merge Sort

Function **mergesort**. Sort a list of n numbers

```
• if the list contains just one item, return it
• sort, using mergesort, the first half of the list
• sort, using mergesort, the second half of the list
```

- merge the two sorted lists together

Quicksort

Function **quicksort**. Sort a list of n numbers

- if the list contains one item or fewer, return it
- pick a *pivot*, e.g., the first item in the list
- put all the values that are less than the pivot into list A, and all the values that are greater than the pivot into list B
- sort, using **quicksort**, list A
- sort, using **quicksort**, list B
- output list A, the pivot, list B

Tree insert

Function **inserttree**. Insert a value in a tree.

- if the tree is empty set the tree to be this node and return it
- if the value is less than the root value then:
- if the left subtree is empty then set the left subtree to be a new node containing the value
- else insert, using **inserttree**, the value in the left subtree
- else if the right subtree is empty then set the right subtree to be a new node containing the value
- else insert, using **inserttree**, the value in the right subtree

Tree sort

Function **treesort**. Print values in a tree in increasing order.

- For each value
- insert the value in the tree using **inserttree**
- do an in-order traversal of the tree to get the sorted data

Heapsort

Function **heapsort**.

Phase 1:

- For each value
- insert value at next leaf
- bubble up the tree, i.e., while value is less than its parent, swap it with its parent.

Phase 2:

- Repeat
- output value at root
- remove value at last leaf and place at root
- bubble down the tree, i.e., while value is greater than a child, swap it with that child. If value is greater than both children, swap it with the *smaller* child.

Binary search

Function **binarysearch**. Find a value v in an array of length n.

- Do **binarysearchrange** for v in the range 0 to $n-1$

Function **binarysearchrange**. Find a value v in an array between indices l and r

- let $m=(l+r)/2$ and look at the value $A[m]$
- if $A[m]=v$ return it
- if $v<A[m]$ return **binarysearchrange** for v in the range l to $m-1$
- if $v<A[m]$ return **binarysearchrange** for v in the range $m+1$ to r

Tree search

Function **treesearch**. Look for a value v in a tree.

- if the tree is empty, return “not found”
- if the value at the root is v, return it.
- if the value at the root is bigger than v, return the result of searching the left subtree using **treesearch**
- return the result of searching the right subtree using **treesearch**

String search

Function **stringsearch**. Look for a pattern P in a text T. Let P have length m and T length n.

- for $i=0$, $j=0$ while $i<m$ and $j<n$ (i says how far along the pattern, j says how far along the text)

```
•     if $P[i] = T[j]$ then
•         $i = i + 1$, $j = j + 1$ (move along both pattern and text)
•     else
•         $j = j - i + 1$, $i = 0$ (reset in text and reset pattern)
• if $i = m$ then return “found at $j - m$” else return “not found”
```