Searching & Sorting

1 Sorting

- Why sort?
- What are some common sorting algorithms?

1.1 Search

The answer to "Why sort?"
(also organization, but *why organize?*)

Common Search Techniques

- Linear Search (aka "Brute Force" Search)
- Binary Search
- Hash tables/Dictionaries

1.1.1 Linear Search

```
1  /*  LinearSearch
2
3      Searches for the value, val, in an array.
4      If successful, the index is returned.
5      If unsuccessful, -1 is returned.
6   */
7  int LinearSearch( const int A[], const int nA,
8                    const int val )
9  {
10     for( int i = 0 ; i < nA ; i++ )
11     {
12         if( A[i] == val )
13             return i;
14     }
15
16     return  -1;
17 }
```

Big $\mathcal{O}$?

1.1.2 Binary Search

```
1  /*  BinarySearch
2
3    Searches for the value, val, in a sorted
4    array (values must be low to high) iteratively.
5    If successful, the index is returned.
6    If unsuccessful, -1 is returned.
7   */
8  int BinarySearch( int A[], int nA, int val )
9  {
10     int mid, low = 0, high = nA-1;
11
12     while( low <= high )
13     {
14         mid = (low+high)/2;
15         if( val > A[mid])
16             low = mid + 1;
17         else if( val < A[mid] )
18             high = mid - 1;
19         else
20             return mid;
21     }
22
23     return -1;
24 }
```

Big $\mathcal{O}$?

1.1.3 Recursive Binary Search

```
1  /*  BinarySearchR
2
3    Recursive search for the value, val, in a
4    sorted array (values must be low to high).
5    If successful, the index is returned.
6    If unsuccessful, -1 is returned.
7   */
8  int BinarySearchR( int A[],
9                     int lo, int hi, int val )
10 {
11    if( hi >= lo )
12    {
13       int m = (lo + hi)/2;
14
15       if( A[m] == val )
16          return m;
17       else if( A[m] > val )
18          return BinarySearchR(A, lo, m-1, val);
19       else
20          return BinarySearchR(A, m+1, hi, val);
21    }
22
23    return -1;
24 }
```

Big $\mathcal{O}$?

1.2 Common Sorting Techniques

- Bubble Sort
- Insertion Sort
- Selection Sort
- Quick Sort
- Merge Sort

Bubble sort Iterate through the array examining adjacent pairs of elements. If necessary, swap them to put them in the desired order (*Brick Sort*).

Insertion sort Iterate through the array placing the ith element with respect to the $i-1$ previous elements.

Selection sort Iterate through the array putting the ith smallest element in the ith location.

Quick sort Partition the list into smaller lists such that every element in the left partition is less than or equal to the right partition. Repeat Quicksort process on the partitions—frequently done using recursion.

Merge sort Useful for sorting very large amounts of data. The basic algorithm:

1. Split the file into smaller files.
2. Sort the smaller files
3. Merge the sorted files

1.3 Sorting Properties

- Comparisons
- Swaps
- Run time

1.4 Selection Sort

Given an array A with N items stored in it.

Loop over each element (i)

Put the smallest first (ith location) each iteration.

`IndexOfSmallest( A, iStart, iEnd )` returns the index of the smallest element in array `A` between (and including) positions `iStart` and `iEnd`.

```
1  void  SelectionSort( aType A[], int nA )
2  {
3      int iSmallest;
4
5      for( i = 0 ; i < nA ; i++ )
6      {
7          iSmallest = IndexOfSmallest( A, i, nA-1 );
8          Swap( A[i], A[iSmallest] );
9      }
10 }
```

Big $\mathcal{O}$?

```
1  int IndexOfSmallest( aType A[], int iStart, int iEnd )
2  {
3      int     index = -1;
4      aType   aMin = A[iStart];
5
6      for( i = iStart ; i <= iEnd ; i++ )
7      {
8          if( A[i] < aMin )
9          {
10             aMin = A[i];
11             index = i;
12         }
13     }
14
15     return index;
16 }
```

Note: Setting index to -1 can be a problem!

Output:

```
nA: 8
Initial array contents:
[ 7, 13, 1, 3, 10, 5, 2, 4 ]
------------------------
In SelectionSort():

IndexOfSmallest:  2
 Pass: 0     [ 1, 13, 7, 3, 10, 5, 2, 4 ]
IndexOfSmallest:  6
 Pass: 1     [ 1, 2, 7, 3, 10, 5, 13, 4 ]
IndexOfSmallest:  3
 Pass: 2     [ 1, 2, 3, 7, 10, 5, 13, 4 ]
IndexOfSmallest:  7
 Pass: 3     [ 1, 2, 3, 4, 10, 5, 13, 7 ]
IndexOfSmallest:  5
 Pass: 4     [ 1, 2, 3, 4, 5, 10, 13, 7 ]
IndexOfSmallest:  7
 Pass: 5     [ 1, 2, 3, 4, 5, 7, 13, 10 ]
IndexOfSmallest:  7
 Pass: 6     [ 1, 2, 3, 4, 5, 7, 10, 13 ]
IndexOfSmallest:  -1
 Pass: 7     [ 1, 2, 3, 4, 5, 7, 10, 13 ]

SelectionSort() finished
------------------------
Final array contents:
[ 1, 2, 3, 4, 5, 7, 10, 13 ]
```

Big $\mathcal{O}$?

```
1  void  SelectionSort( aType A[], int nA )
2  {
3      int iSmallest;
4
5      for( int i = 0 ; i < nA ; i++ )
6      {
7          //iSmallest = IndexOfSmallest( A, i, nA-1 );
8
9          aType  aMin = A[i];
10
11         for( int j = i ; j <= nA-1 ; j++ )
12         {
13             if( A[j] < aMin )
14             {
15                 aMin = A[j];
16                 iSmallest = j;
17             }
18         }
19
20         Swap( A[i], A[iSmallest] );
21     }
22 }
```