4.3.4 Search & Sorting Algorithms

Linear Search

Description

Each item is sequentially compared to the search item until the item is found or the end of the array/list is reached.

Example Code

```
function linearSearch(aList, searchItem)
      index = -1
      i = 0
      found = False
      while i < aList.length and found = False
            if aList[i] == searchItem then
                  index = i
                  found = True
            endif
            i = i + 1
      endwhile
      return index
endfunction
```

Time Complexity

Best Case: O(1)

Average Case: O(n)

Worst Case: O(n)

Space Complexity

O(1)

Binary Search

Description

The item at the midpoint is compared to the search item. If they are equal then the search item was found. If the search item is smaller than the midpoint then the left sublist is searched. If the search item is larger than the midpoint then the right sublist is searched.

This is repeated until the search item is found or there are no more items.

Precondition

The list/array must be sorted before the binary search can be used.

Example Code

```
function binarySearch(aList, searchItem)
      index = -1
      found = False
      first = 0
      last = aList.length - 1
      while first <= last and found = False
            mid = (first + last) DIV 2
            if aList[mid] = searchItem then
                  index = mid
                  found = True
            else if aList[mid] < searchItem then
                  first = mid + 1
            else
                  last = mid – 1
            endif
      endwhile
      return index
endfunction
```

Time Complexity

Best Case: O(1)

Average Case: O(logn)

Worst Case: O(logn)

Space Complexity

O(1)

Bubble sort

Description

A pass is made through the array/list comparing each item with the one next to it. If they are in the wrong order swap them.

Continue to make passes until a pass is made that makes no swaps.

Example Code

```
procedure bubblesort(aList)
      swapped = True
      while swapped == True
            swapped = False
            for i = 0 to (aList.length - 2)
                  if aList[i] > aList[i+1]
                        temp = aList[i]
                        aList[i] = aList[i+1]
                        aList[i+1] = temp
                        swapped = True
                  endif
            next i
      endwhile
endprocedure
```

Time Complexity

Best Case: O(n)

Average Case: $O(n^2)$

Worst Case: $O(n^2)$

Space Complexity

O(1)

Merge sort

Description

The array/list is split into left and right sublists. These sublists are repeatedly split until each item is on its own. The smaller lists are then merged back together in order. This is repeated until the whole list is merged into one sorted list.

Example Code

```
procedure mergesort(aList)
      if aList.length > 1
            mid = aList.length DIV 2
            left = aList[:mid]
            right = aList[mid:]
            mergesort(left)
            mergesort(right)
      endif
      i = 0
      j = 0
      k = 0
      while i < len(left) and j < len(right)
            If left[i] < right[j]
                  aList[k] = left[i]
                  i = i + 1
            else
                  aList[k] = right[j]
                  j = j + 1
            k = k + 1
            endif
      endwhile
      while i < len(left)
            aList[k] = left[i]
            i = i + 1
            k = k + 1
      endwhile
      while j < len(right)
            aList[k] = right[j]
            j = j + 1
            k = k + 1
      endwhile
endprocedure
```

Time Complexity

Best Case: O(nlogn)

Average Case: O(nlogn)

Worst Case: O(nlogn)

Space Complexity

O(n)

Visualisation