C++
Algorithms

CO658 Data Structures & Algorithms

Topics

- Swap function
- Sort Algorithms
- Search Algorithms

Swap

- Used by a number of sort algorithms to swap two values.
- In the examples and exercises it is used to swap integer values.

```
function Swap (out int a , out int b)
        temp = b
        b = a
        a = temp
end function
```

- The function has no return type
- Parameters are passed by reference or address

Swap Implementation

- The implementation below uses pointers as parameters.
- This allows the arguments to be changed within the function.

```
void Swap(int * a, int * b){
		int temp = *b;
		*b = *a;
		*a = temp;
}
```

Sort Algorithms

- An algorithm that orders data
- In our examples we will be ordering arrays of integer values.
- Sorted data is easier to search
- Especially array data which can be randomly accessed as opposed to a Linked List which is sequentially searched.
- The efficiency of the algorithm can be measured using the Big O notation.

C++
Big O Notation

CO658 Data Structures & Algorithms

Asymptotic Complexity

- The run-time of each algorithm will depend on several factors:
  - The number and type of operations performed by the algorithm
  - The number of elements or nodes in a data structure
  - How the elements or nodes are arranged (sorted or unsorted)
  - Also dependent upon other factors such as computer architecture and processor speed

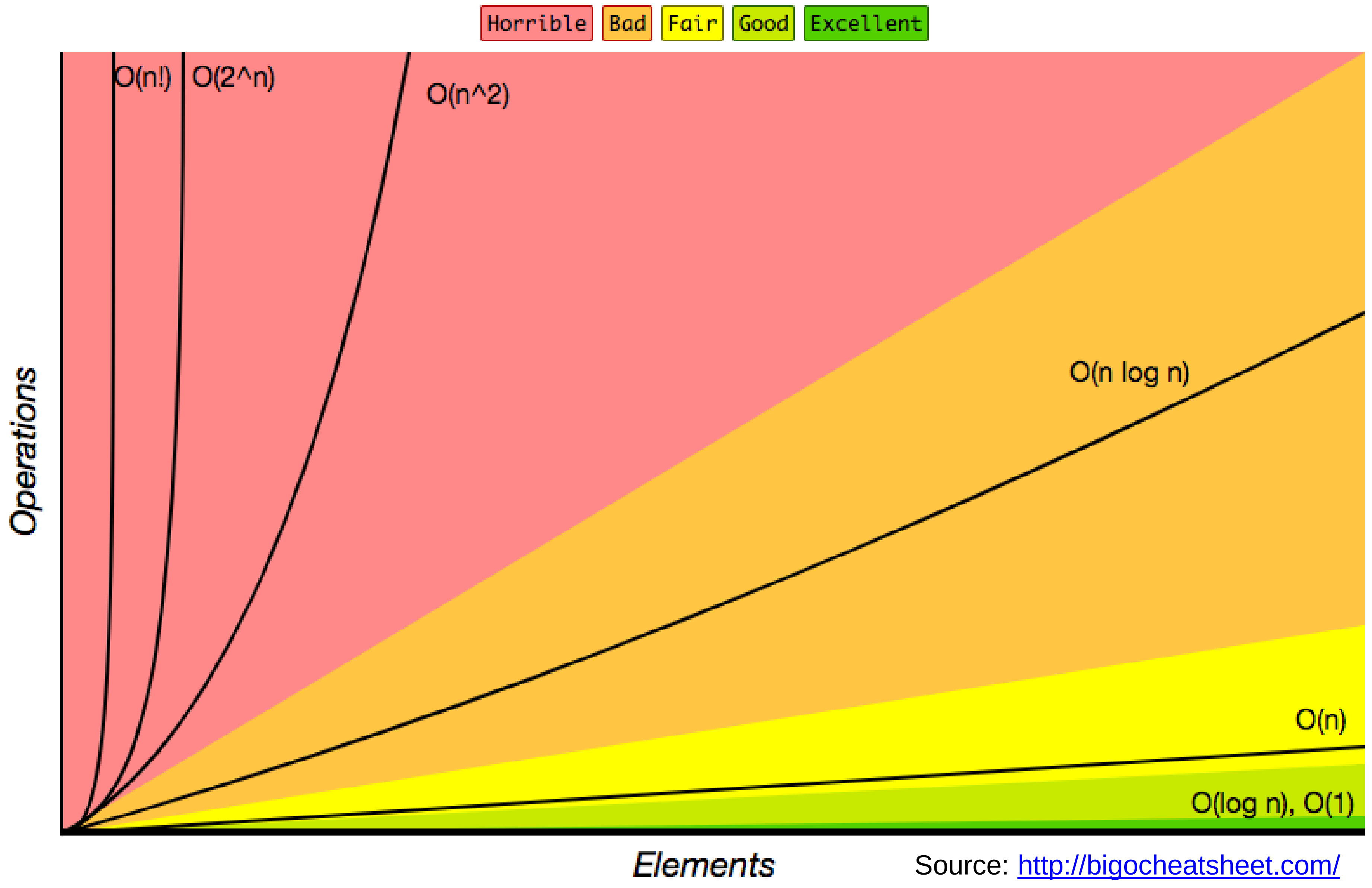
Fast Big O runtime calculations

- O(1) is ‘constant time’
  - Operations (such as Insert or Delete in some data structures) take the same time regardless of how many elements or nodes there are (n)
- O(n) is ‘linear time’
  - The runtime of the algorithm is directly proportional to n
  - One loop that prints each element of the data structure would take O(n)
- O(log n) is ‘logarithmic time’
  - Runtime only increases fractionally as n increases
  - $\log_2 (8) = 3$
  - $\log_2 (8000) = 12.96$
- O(n log n) is ‘linearithmic time’
  - Runtime is O(log n) x n
  - Divide and conquer algorithms work on this principle

Slow Big O runtime calculations

- $O(n^2)$ is ‘quadratic time’
  - Runtime will geometrically increase as n increases:
  - $5^2 = 25$
  - $10^2 = 100$
  - Nested loops usually take $O(n^2)$ if the data structure is iterated through for every element of the data structure
- $O(n!)$ is ‘factorial time’
  - Runtime will exponentially increase as n increases (but by larger intervals)
  - $5! = 120$
  - $10! = 3,628,800$
  - Brute force algorithms that try all combinations… very slow…

Big-O Complexity Chart



Source: http://bigocheatsheet.com/

Best, worst and average cases

- Best case
  - If we’re searching for an element that happens to be near the start of the data structure or the elements are nearly sorted; the algorithm will complete quickly
- Worst case
  - If we’re searching for an element happens to be near the end of the data structure or the data structure is completely unsorted, the algorithm will take longer
- Average cases
  - If the element resides around the middle of the array

Tips for calculating time

1) Add time for each step
   - Nested loop: O(n) for each loop, therefore: O(n) x O(n) = O(n^2)
2) Drop constants
   - Two loops that perform sequentially would not be O(2n) but O(n)
   - We're interested in distinguishing between categories such as: constant, linear, quadratic, logarithmic, factorial etc
3) Different inputs should be written with different letters
   - If we're using a nested loop to iterate through two different arrays (a) and (b) it's not O(n^2), but O(a x b)
4) Drop non-dominant terms
   - O(n + n^2) should become O(n^2) because we are only interested in categories such as constant, linear, quadratic, logarithmic

C++

Sorting algorithms

CO658 Data Structures & Algorithms

Selection Sort

- Repeatedly locate the smallest unprocessed element in the array and position it in its correct index location.
- Efficiency average and worst O(N^2).

ex: Sort an array {5,1,3,4,2} using Selection Sort.

Locate the smallest element 1 and swap it with the first element

{1,5,3,4,2}

Locate the smallest element from the sub set, starting from after the last sorted element.

{5,3,4,2} and swap with the first – gives { 2,3,4,5 }

Giving {1,2,3,4,5}

Selection Sort

```
function SelectionSort (array a, int size)
    for n := 0 to size-1 do
        smallest = n
        for i := n+1 to size-1 do
            if a[i] < a[smallest]
                smallest = i
            end if
        end loop
        Swap(a[n],a[smallest])
    end loop
end function
```

Activity

Attempt exercise 1, 2 and 3:

- Implement Selection Sort and output the time taken to sort different array sizes

Bubble Sort

Make repeated passes through a list of items, exchanging adjacent items if necessary.

At each pass, the largest unsorted item will be pushed in its proper place.

Efficiency Average and worst O(N^2)

Bubble Sort Explanation

Compare pairs of adjacent elements of the sequence, if they are in the wrong order swap them and do this till there are no more swapping's to do.
ex: Sort an array {5,3,1,4,2} using Bubble Sort.

Compare first two, as 5 > 3, they are swapped

- {3,5,1,4,2} Again compare next two, as 5 >1, they are swapped
- {3,1,5,4,2} Keep swapping
- Until the first pass is complete {3,1,4,2,5}
- Second Pass : compare 3 & 1, as 3 > 1 they are swapped
- {1,3,4,2,5} compare 3,4 they are in correct order
- Compare 4 & 2 and swap compare 4 & 5 no swap.
- So far we have { 1,3,2,4,5}
- Final Pass : Compare 1 & 3 no swap. Compare 3&2 and swap.

Bubble Sort Algorithm

```
function BubbleSort (array a, int size)
    for i := 0 to < size do
        for j := 0 to < size-1 do
            if a[j] > a[j+1]
                Swap(a[j],a[j+1])
            end if
        end loop
    end loop
end function
```

Enhanced Bubble Sort

- Need to modify algorithm to exit when no change made in inner loop.
- Efficiency O(n^2)
- Even if the array is sorted prior to being passed to the algorithm, the operation will take O(N^2).

Enhanced Bubble Sort

```
function BubbleSort (array a, int size)
    sorted = false
    lastUnsorted = size -1
    while sorted is false
      sorted = true
      for j := 0 to < lastUnsorted do
          if a[j] > a[j+1]
              Swap(a[j],a[j+1])
              sorted = false
          end if
      end loop
      decrement lastUnsorted
    end loop
end function
```

Activity

Attempt exercise 4 and 5:

- Implement Bubble Sort and Enhanced Bubble Sort

Search Algorithms

- Key is the value being searched for
- Some algorithms assume the data is sorted
- The search function returns the index of the key within the array if found
- Returns -1 if the key is not found

Sequential Search

- Also known as linear search
- Each element is checked in sequence until a match is found.
- Efficiency O(n)
- Only suitable for small data sets

Sequential Search

```
int function SequentialSearch (array A, int size, int  key)
        loop  i = 0 to size -1
                if A(i) = Key then
                        return i
                end if
        end loop
        return -1
end function
```

Binary Search

- Searches sorted data only
- The key is compared to the middle value if not matched and the key is less than the value the action is repeated on the left portion of the array otherwise the right.
- Efficiency O(log(n))
- Suitable for large data sets

Binary Search

```
int function BinarySearch (array a, int size, int  key)
        middle = 0,  lower = 0, upper = size - 1
        loop
                middle = (lower + upper) / 2
                if (key < a[middle] then
                        upper = middle – 1
                else if (key > a[middle) then
                        lower = middle + 1
                else return middle
                if lower > upper return -1
        end loop
End function
```

Activity

Attempt exercise 6 and 7:

- Implement Linear and Binary Search
- Q: Which is faster?

Summary

- While these examples use arrays of integers it is more likely in reality that the arrays would contain pointers to objects that have a data member that determines their position in the array.
- I’ve deliberately not included the code so that you can implement the algorithms yourself within the practical.