Introduction to Data Structures

Credits

- Some of these slides were written by me
- Some information was found on the internet!
- Some information was from Michael Main, Data Structures and Other Objects Using Java
- Some information was from Wikipedia

Today's Goal

- Talk about standard data structures and where they are used
- Talk about sorting and searching
- Talk about running time analysis

Data Structures Overview

- Primitive Types
  - Bit, byte, etc.
- Collections
  - Bags
  - Array Lists (Sequences)
  - Linked Lists (Sequences)
- Abstract Data Types
  - (Interfaces, Abstract Classes)
- Stacks – Last in, first out
- Queues – First in, first out
- Recursion
- Trees
  - Binary Search
  - Heaps
- Searching
  - Serial/Linear Search
  - Binary Search
  - Hashing
- Sorting
  - Quadratic Sorts
  - Quick Sort
  - Merge Sort
  - Heap Sort
- Running Time Analysis
  - Big-Oh

Bit

- Tiniest data structure
- On or Off
- 1 or 0
- Fundamental building block of all data

“Byte”

- 8 bits
- Bytes store numbers
- We use numbers to represent everything else
- Numbers are stored, ultimately in binary
- Example:
  00000001 is 1
  00000100 is 4
  00001100 is 12

Nybble / nyble/nibble

- Half a byte, 4 bits

Byte

- A byte can contain a number
- A byte can contain a character
- Byte = 25;
- Byte = ‘A’
- What if we want a word?

Array of bytes

char bytes[5] ;

bytes[0] = ‘M’

Bytes[1] = ‘I’

Bytes[2] = ‘K’

Bytes[3] = ‘E’

Bytes[4] = 0

(Remember , computers like to start counting from zero)

Bigger Numbers

- Integers
- What if we want a number bigger than the largest value a byte can hold
- Integer types:
  - Short - 2bytes (packaged as 1)
  - Int - (4 bytes packaged as 1)
  - Long (8 bytes packaged as 1)

Review

- We understand multi byte arrays for storing words (array)
- We understand multi byte structures for storing numbers (types of int)
- We can have an array of ints too

One Dimensional Array of integers

Int ints[5] ;

ints[0] = 3

ints[1] = 4

ints[2] = 1

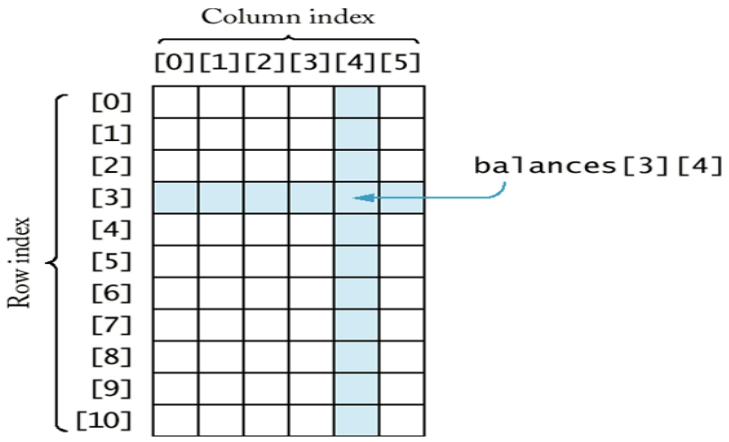
ints[3] = 2

ints[4] = 0

So what is int #2?

What is int #4?

Two Dimensional Array



Array of words

```
return $stopwords = array("a", "about", "above", "above", "across", "after", "afterwards", "again", "against",
"all", "almost", "alone", "along", "already", "also","although","always","am","among", "amongst", "amoungst", "amount",
"an", "and", "another", "any","anyhow","anyone","anything","anyway", "anywhere", "are", "around", "as", "at", "back","be",
"became", "because","become","becomes", "becoming", "been", "before", "beforehand", "behind", "being", "below", "beside",
"besides", "between", "beyond", "bill", "both", "bottom","but", "by", "call", "can", "cannot", "cant", "co", "con",
"could", "couldnt", "cry", "de", "describe", "detail", "do", "done", "down", "due", "during", "each", "eg", "eight",
"either", "eleven","else", "elsewhere", "empty", "enough", "etc", "even", "ever", "every", "everyone", "everything",
"everywhere", "except", "few", "fifteen", "fify", "fill", "find", "fire", "first", "five", "for", "former", "formerly",
"forty", "found", "four", "from", "front", "full", "further", "get", "give", "go", "had", "has", "hasnt", "have", "he",
"hence", "her", "here", "hereafter", "hereby", "herein", "hereupon", "hers", "herself", "him", "himself", "his", "how",
"however", "hundred", "ie", "if", "in", "inc", "indeed", "interest", "into", "is", "it", "its", "itself", "keep", "last",
"latter", "latterly", "least", "less", "ltd", "made", "many", "may", "me", "meanwhile", "might", "mill", "mine", "more",
"moreover", "most", "mostly", "move", "much", "must", "my", "myself", "name", "namely", "neither", "never", "nevertheless",
"next", "nine", "no", "nobody", "none", "noone", "nor", "not", "nothing", "now", "nowhere", "of", "off", "often", "on",
"once", "one", "only", "onto", "or", "other", "others", "otherwise", "our", "ours", "ourselves", "out", "over", "own",
"part", "per", "perhaps", "please", "put", "rather", "re", "same", "see", "seem", "seemed", "seeming", "seems", "serious",
"several", "she", "should", "show", "side", "since", "sincere", "six", "sixty", "so", "some", "somehow", "someone",
"something", "sometime", "sometimes", "somewhere", "still", "such", "system", "take", "ten", "than", "that", "the",
"their", "them", "themselves", "then", "thence", "there", "thereafter", "thereby", "therefore", "therein", "thereupon",
"these", "they", "thicky", "thin", "third", "this", "those", "though", "three", "through", "throughout", "thru", "thus",
"to", "together", "too", "top", "toward", "towards", "twelve", "twenty", "two", "un", "under", "until", "up", "upon", "us",
"very", "via", "was", "we", "well", "were", "what", "whatever", "when", "whence", "whenever", "where", "whereafter",
"whereas", "whereby", "wherein", "whereupon", "wherever", "whether", "which", "while", "whither", "who", "whoever",
"whole", "whom", "whose", "why", "will", "with", "within", "without", "would", "yet", "you", "your", "yours", "yourself",
"yourselves", "the");
```

One characteristic of arrays

- Array Lookup is fast!
- You know the index, you go right to the ‘box’.

Think of mail boxes

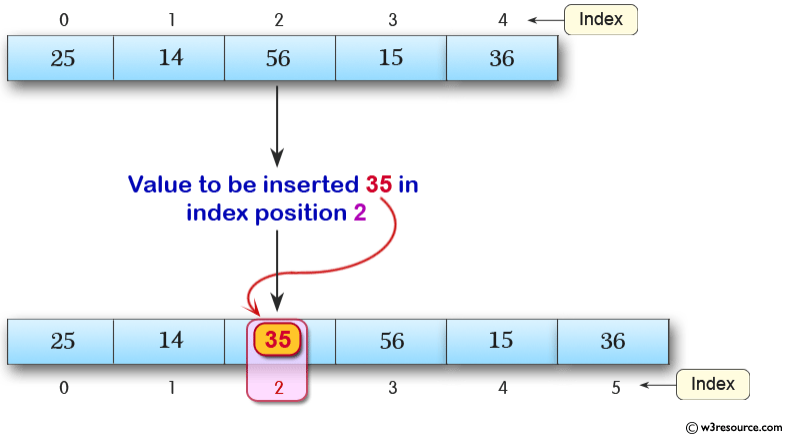
Collection

- A collection is a grouping of data items that have some sort of relation or shared significance.
- Array is one data structure that can be used for a collection
- An array has the benefit of fast lookups (If you know where to look!)

Array Weakness

- Suppose we want to insert into the middle of an array?
- What happens?

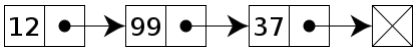
Insert New Value into Array

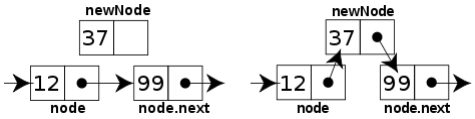


https://www.w3resource.com/java-exercises/array/java-array-exercise-9.php

Alternative

- Linked List
- Every node References the Next node





https://en.wikipedia.org/wiki/Linked_list

Linked Lists

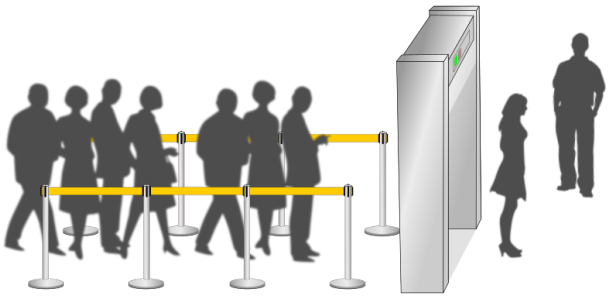
- Can be used for a collection
- Quicker than array to add data anywhere
- Slow to lookup as you need to search all items.
- Good for applications that get a lot of new data, but look data up infrequently
- Or applications that only look at the front or back of the list.

Array vs. Linked List

- Array: Good for fast random access lookups
- Array: Slow for adding new data to keep it sorted
- Linked List: Fast for adding new data anywhere.
- Linked List: Slow'ish for finding data

Queues

- First in, First out data structure



Queues

- First item added to a queue is the first item removed.
- Can be built from an array or a linked list

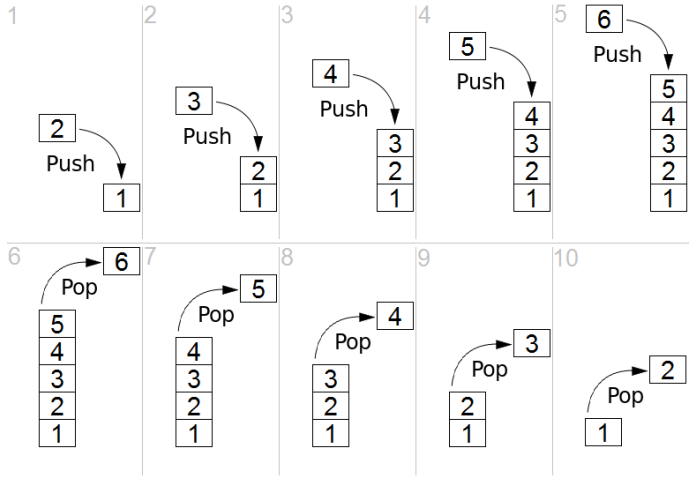
Queues

- Where are queues used?
- Imagine a web-site that services files to thousands of users. You can't service them all at once, so you serve first come first serve!
- Thanks stackoverflow.com!
- Used heavily by operating systems
  - Devices queues, process queues,

Stacks

- Last in, first out data structure
- Can also be built with array or linked list
- Where used?
  - Undo in word! (or anything)
- Extensive operating system use

Stack



https://en.wikipedia.org/wiki/Stack_(abstract_data_type)

Summary so far

- Bits
- Bytes
- Ints/Longs/Shorts
  - These are all primitive types
- Arrays
- Linked List
- Queues
- Stacks

Abstract Data Types

- We know what a data type can do
- How it is done is hidden for the user
- Put another way
  - users are not concerned with how the task is done but rather with what it can do.
- Stack and Queue are examples of abstract data types

Queue

- Enqueue
- Dequeue
- Remove

Stack

- Push
- Pop
- Empty

Information Hiding

- Information Hiding
  - Only expose what is necessary to use the class/object
  - Instance variables should all be private
  - You should use ‘getter’ and ‘setter’ methods to manipulate these values.
  - This is important because:
    - You can validate the values before storing them.
    - Your methods that use the instance data can be sure they are correct.
    - Remember, the author of a method is responsible for it not ‘crashing’.

Running Time Analysis

- Algorithm Running Analysis

Running time analysis

- Operational Ceiling
  - The ceiling or limit at which the application can operate before failure.
- Worst Case
  - The maximum number of operations
- Best Case
  - Is the best we can hope to do!
  - The minimum number of operations required
- Average Case
  - Average amount of operations required.

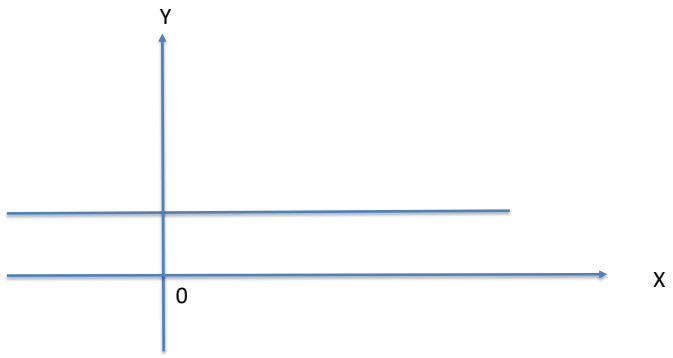
Running time analysis (why?)

- Does the algorithm work fast enough for my needs?
- How much might changing the size of input affect my performance?
- Is one algorithm better than another?

Big-Oh notation (constant)

- **Constant time**
  - **O(1)**
  - Operation takes the same amount of time no matter what n is.
- For example:
  - Reference a location in an array.
  - A = i[20]
  - A = i[35];
  - Takes as much time to reference the spot in the array, no matter how big the array i[] is.

Constant Time



Flat Line!
0 Slope!

Big-Oh Notation (logarithmic)

- Remember "Logarithms"
- *$y=b^x$ means $\log_b(y) = x$*

  Examples for $\log_{10}$:
  - $\log_{10} 10 = 1$
  - $\log_{10} 100 = 2$
  - $\log_{10} 1000 = 3$
  - $\log_{10} 10000 = 4$

Big-Oh Notation (logarithmic)

- Logarithmic Time
- O(log n)
- If the 'n' you are changing falls into bands, it is likely logarithmic time. For example:
  - Your n changes from 10 to 15 and you still need 1 operation but when your 'n' changes from 15 to 200 you need 2 operations.
- Example: Calculate Number of digits in a number
  - 913 3
  - 233 3
  - 32 2

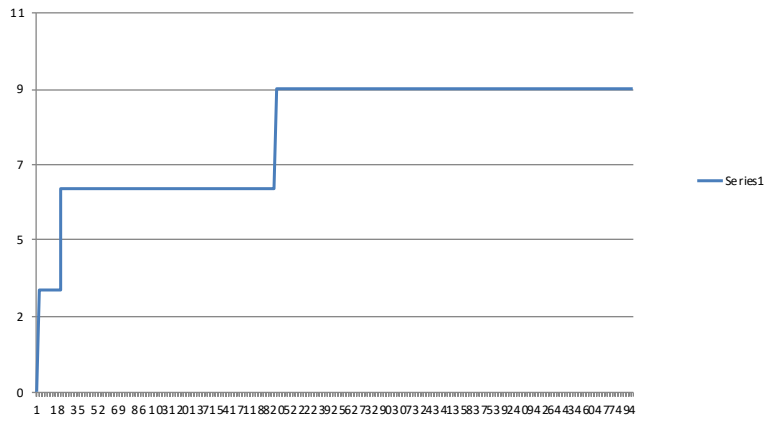
How many times can you divide and have the number still be greater than 1

```
public class Divide {
    public static void divideIt(int number)  {
        int temp = 0;
       timer time=startTime();

        while (number > 1)          // 1 - compare
          {
            number = number / 10; // 1 assignment, 1 divide
          }

         timer time2=EndTime();

    }

     System.out.print(number+","+EndTime-StartTime+",");

}
```

How many times can you divide and have the number still be greater than 1

```
public class Divide {
    public static void divideIt(int number)  {
        int temp = 0;
       timer time=startTime();

          something_that_I_cant_even_see(number);

         timer time2=EndTime();

    }

     System.out.print(number+","+EndTime-StartTime+",");

}
```

Plotted Output



Big-Oh Notation (Linear)

- Linear Time
- O(n)
- Requirements:
  - Each time 'n' changes the operations count changes.
  - When plotted the operations count follows a straight line.

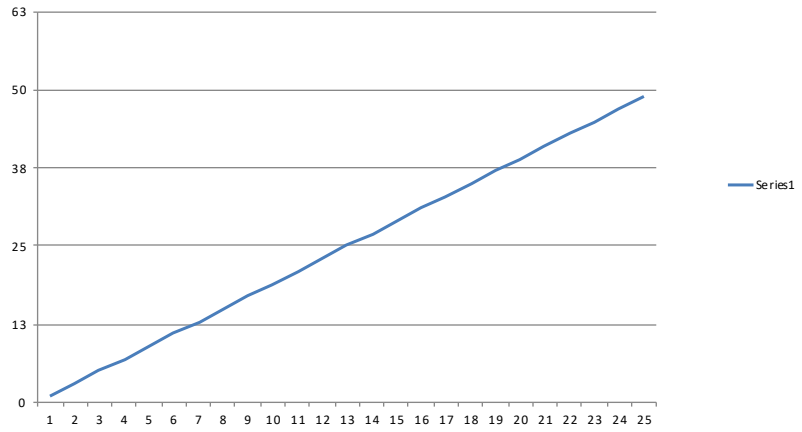
Linear example:

```
public static double doSomethingLinear(double myN)
 {
       timer time=startTime();

      // Generally one loop is linear
       for (i=0;i<myN;i++)
       {
        retVal = retVal * myN;
       }

       timer time2=EndTime();

       System.out.print(myN+","+EndTime-StartTime+",");
       return retVal;
 }
```

Sample Data

- 0,1
- 1,3
- 2,5
- 3,7
- 4,9
- 5,11
- 6,13
- 7,15
- 8,17
- 9,19
- 10,21
- 11,23
- 12,25
- 13,27
- 14,29
- 15,31
- 16,33
- 17,35
- 18,37
- 19,39
- 20,41

Picture



Line with some slope

Big-Oh notation (Quadratic)

- Quadratic Time
- $O(n^2)$
- Requirements:
  - Each time 'n' changes the operations count changes…BUT
  - When plotted the operations count DOES NOT follow a straight line, instead it shoots up rapidly.

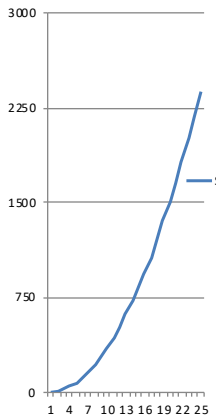
Quadratic Example

```
public static long profound_thing_do (long count)
{
 int i,j; long retVal = 0;  int temp=0;

    time1 = startTime();

     // generally 2 loops are quadratic
    for (int i=0;i<count;i++)
      for(j=0;j<count;j++)
           do_something_profound(count);

   time2 = EndTime()
   System.out.println(myN+","+time2-time1);
   return retVal;
}
```

Possible Data

- 0,1
- 1,8
- 2,23
- 3,46
- 4,77
- 5,116
- 6,163
- 7,218
- 8,281
- 9,352
- 10,431
- 11,518
- 12,613
- 13,716
- 14,827
- 15,946
- 16,1073
- 17,1208
- 18,1351
- 19,1502
- 20,1661

Graph, when plotted Quadratic



Remember — Running Time Analysis

- Don't try to count instructions
- Put 'counters' inside your loops
- Change 'n'
- Run
- Plot with excel

REMEMBER:Running time analysis

- Operational Ceiling
  - The ceiling or limit at which the application can operate before failure.
- Worst Case
  - The maximum number of operations
- Best Case
  - Is the best we can hope to do!
  - The minimum number of operations required
- Average Case
  - Average amount of operations required.

End of Part #1

- Data types
- Arrays
- Linked Lists
- Queues
- Stacks
- Algorithm Complexity
- Half a semester in 65 slides

Searching

- Searching is one of the most important operations performed
- Having elements in order can be a big help.
- Linear search
- Binary search
- Hash table

Linear Search/ Serial Search

- Serial Search or Linear Search
  - Linear time
- Searches for desired element starting at first element.
- Goes one element at a time until it is found.
- Benefits
  - List does not have to be ordered
  - Easy to write
- Drawback: O(n) time lower bound

Linear Searching – for example

- For (i=0;i<n;i++)
  - (If data[i] == value_I_am_looking_for)
    - break;
- Easy to see 'Lower Bounds' would be linear
  - **O(n)**
- Best case really fast!
- Average case better than worst case..

Binary Search

- Divide and conquer search
- **Lower bounds, if done properly, is big-oh**
  - **O($\log_n$) Logarithmic!!!**
- Worst case?
  - Element not found
    - Still faster than linear search
- Best case
  - Element found in center of list

Binary Search Pseudocode

Pseudocode:

If (range contains only one element):

    Look for desired value

Else

    Get midpoint of range

    Is this my element?

    Determine which half of range contains the value

    'binary search' that half of the range

Binary Search Example

- Find 37
- Start: 9,11,13, 14, 27, 37, 39,40, 41
  - *9 elements*

Possible Exam Question

- Find 9
- Start: 9,11,13, 14, 27, 37, 39,40, 41
- Assume middle = first + size/2
- What is the first element searched?
- What is the second element searched?
  - Start: *9,11,13, 14*, 27, 37, 39,40, 41
- Write the order in which each element is examined.

Hashing Slides (Some from)

- www.mis.nsysu.edu.tw/~syhwang/Courses/IR/**Hashing.ppt**
- San-Yih Hwang a former student of the U of M.

Hashing

- Hashing is an efficient information retrieval strategy providing access to information based on a key
- Time complexity, performance, the goal is "avoiding collisions"
  - (A collision occurs when two or more keys map to the same location)
- Space complexity, compactness.
  - Basically we can avoid collisions, but waste a lot of space.

Example: Library Card Catalogs

Sample Catalog Record
Author: Kesey, Ken.
Title: One flew over the cuckoo's nest, a novel.
Published: New York, Viking Press [1962]
LC Call No.: PZ4.K42On
Subjects: Psychiatric hospital patients--United States--fiction
Control No.: 62008602

https://en.wikipedia.org/wiki/Library_catalog

Hashing

- Information is stored in a hash table
- Information is accessed using a key
- hash table: an array of locations which can hold data record
  - Sometimes called ***buckets or slots***.
- A hash function maps between a key and a bucket.
  - I.e., it given a key, it tells which bucket to look in for the data

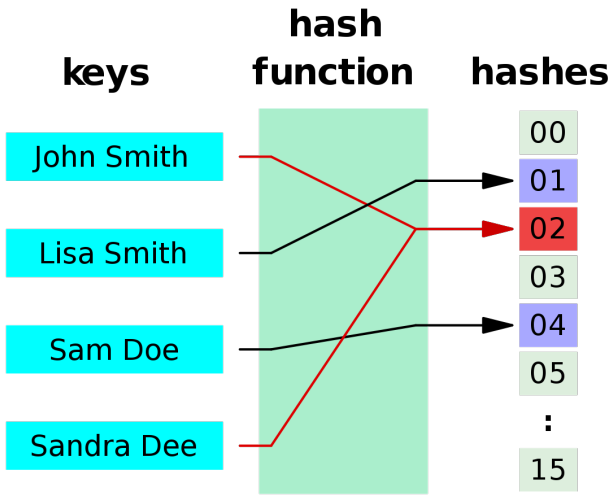
Hashing function

- A **perfect hash function** would distribute data across the buckets such that *no collisions ever occurred*
- Perfect hash functions are hard to find.
- Sometimes a hash function returns the same bucket for two different keys. This is a collision .

Hash Tables

- **Open addressing**
  - Each slot contains a key-value pair (both!)
  - Key-value, because we likely don't have a perfect hash function.
  - If a collision occurs the algorithm calculates another location.
    - perhaps next location

Open address hashing



Review Hash Tables

- Can be O(constant)
- We use a **hash function** to map a 'key' to an integer index. <-- important/remember
- We use generated index to store and lookup data in an array of objects.
- A **perfect hash function** maps elements to distinct integers, with no collisions!

Summary

- Hash tables associate a collection of records with keys.
- The record location is based on the hash value create from the record's key.
- Collisions cause the next available slot is used.
- Searching with the key is then quick.
- When deleting a record the location is marked as deleted so searches skips that spot.

Sorting

Sorting!

- Bubble sort
- Selection sort
- Insertion sort
- Heap sort - nlogn
- Quick sort – n logn
- ~~Merge sort – n logn~~

Bubble Sort

- Compare First Two Numbers
  - Swap if you need to
- Compare Second Two Numbers
  - Swap if you need
- Then start over
- Do that process ’n’ times, where ’n’ is the length of the list.

Sorting

- Bubble sort

```
for (i=n-1; i>=0; i--)
  for (j=0; j<i; j++)
     if (data[j] > data[j+1])
          swap(data[j], data[j+1])
```

- Time complexity $O(n^2)$

Insertion Sort

- Make a new empty, sorted list
- Take each number from the unsorted list
- Put it into the correct position on the sorted listTime complexity $O(n^2)$
- https://en.wikipedia.org/wiki/Insertion_sort

Example

- 64 25 12 22 11

- 64
- 25, 64
- 12,25,64,
- 12,22,25,64
- 11,12,22,25,64

Selection Sort

- Loop
- Find the minimum element
- Put it at the beginning (or the new beginning!)
- Time complexity $O(n^2)$

```
64 25 12 22 11
11 64 25 12 22
11 12 64 25 22
11 12 22 64 25
11 12 22 25 64
```

https://en.wikipedia.org/wiki/Insertion_sort

Selection Sort Example

Sorted sublist	Unsorted sublist	Least element in unsorted list
( )	(11, 25, 12, 22, 64)	11
(11)	(25, 12, 22, 64)	12
(11, 12)	(25, 22, 64)	22
(11, 12, 22)	(25, 64)	25
(11, 12, 22, 25)	(64)	64
(11, 12, 22, 25, 64)	( )	

https://en.wikipedia.org/wiki/Selection_sort

Selection Sort, Example #2 (Swap, not move)

```
64 25 12 22 11
11 25 12 22 64
11 12 25 22 64
11 12 22 25 64
```

Quicksort

- Select a 'pivot' element of the list
  - The ‘magnitudes’ of the elements relative to the pivot
  - Elements less than the pivot are moved left
  - Elements greater than or equal to the pivot are moved to the right
  - At this point all we have to do is sort the left sublist and the right sublist ...
  - Quicksort left, quicksort right!
  - Time complexity $O(n\log_n)$ – normally

Quicksort

- 4,7,9,1,3,2,8
- Pivot: 4 (index is 3)
- 1,3,2,4,7,9,8 <- reorder such that everything less than 4 is left
- Quicksort left <- everything greater is right
  - 1,3,2
  - Pivot 1 (index is 0)
- Quicksort right
  - 7,9,8
  - Pivot 7

So why would you even consider?

- Why consider bubble, selection, or insertion sorts?
  - Very quick to write
  - Stable, no worry about stack size
  - Can sort files on a disk

Bogo Sort – From Wiki

- Based on the generate and test paradigm. It is not useful for sorting, but may be used for educational purposes, to contrast it with other more realistic algorithms
- If bogosort were used to sort a deck of cards, it would consist of checking if the deck were in order, and if it were not, throwing the deck into the air, picking the cards up at random, and repeating the process until the deck is sorted.
- Its name comes from the word *bogus*.

Trees

Problem with lists

- Accessing a item from a linked list takes linear time
- Accessing a item from an array list takes constant time
- Binary trees can improve upon this and take logarithmic time for the average case
  - Worst case degenerates to linear

Tree Vocabulary

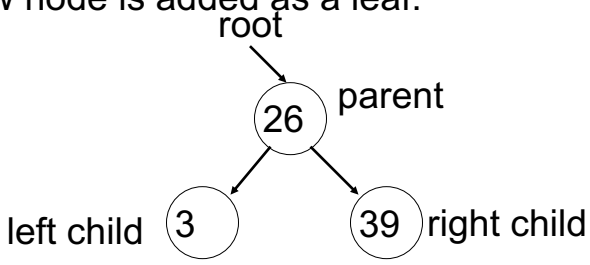
- Remember these…
  - Preorder is root, left, right
  - Postorder is left, right, root
  - In order is left, root, right

Trees

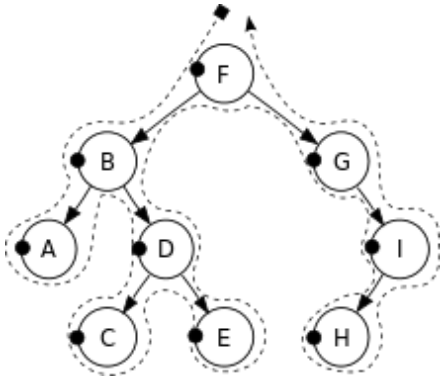
- Binary Tree – each node has at most 2 leaves
  - Not necessarily ordered
- Binary Search Tree – each not has at most 2 leaves and is ordered!

Binary Search Trees

- A binary search tree is a binary tree in which **every node's** left subtree holds values less than the node's value
- Every right subtree holds values greater than the node's value.
- A new node is added as a leaf.

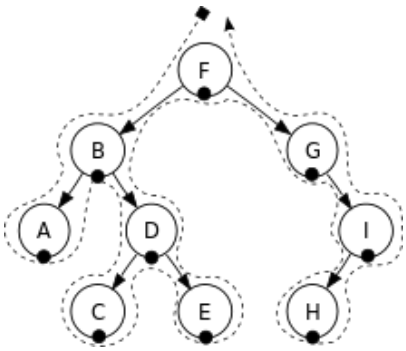


Tree-Traversal Pre-Order



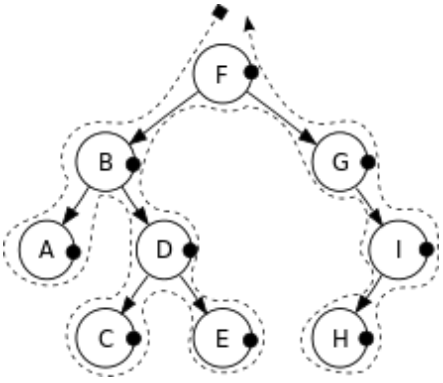
https://en.wikipedia.org/wiki/Tree_traversal

Tree-Traversal In-Order



https://en.wikipedia.org/wiki/Tree_traversal

Tree-Traversal Post Order



https://en.wikipedia.org/wiki/Tree_traversal

Remember. What is the difference?

- Binary Tree
- Binary Search Tree

What would this tree look like?

If we added these values to a binary search tree..

71 22 19 8 10 28 -2

What is the resulting tree?

Tree's

- What is good about pre-order?
  - We can print out a binary search tree and read it back in to build exactly the same binary tree.
- What is good about in-order?
  - The binary ***search*** tree is printed ***in-order***?
  - Can't we make a sort with this? Sure!

Heaps!

Heaps

- A heap is a binary tree in which the elements can be compared with each other using *total order semantics*.
- A **total order semantic means** all the elements of a class can be placed in a single line, proceeding from smaller to larger along the line.
- A binary search tree needs this too

Total Order Semantic

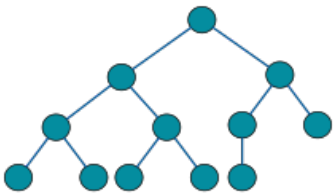
- Equality: (x==y) if they are identical
- Totality: Only one of (x<y) or (x==y) or (x>y)
- Consistency:
  - (x > y) means (y < x)
  - (x!=y) means !(x==y)
  - Blah blah blah
- Transitivity
  - If (x < y) and (y < z) then (x<z)

Heaps – 2 Rules

- The element contained by each node is greater than or equal to the elements of that node's children.
- The tree is a complete binary tree so that every level except the deepest contains as many nodes as possible; at the deepest level, all the nodes are as far left as possible.

Complete Binary Tree

- Every level except the deepest must have as many nodes as possible.
- Deepest level, all nodes are as far left as possible

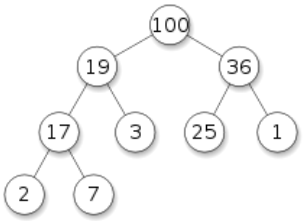


https://en.wikipedia.org/wiki/Binary_tree

98

Heaps

- Objects are organized such that each value is at least as large as its children.
- (Or at least as small depending on how you look at it)
- Heaps are complete trees

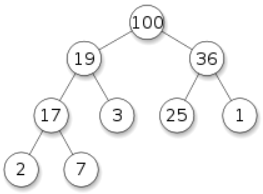


Max Heap Example

https://en.wikipedia.org/wiki/Heap_(data_structure)

Heap Demo

- Heap Demo
- https://youtu.be/WCm3TqScBM8



Heap Practice

Where are heaps used?

- Not for searching data
- Priority queues
  - Priority queue = a queue where all elements have a “priority” associated with them
  - Remove in a priority queue removes the element with the smallest ‘priority’

Heap Sorting

- Put items in a heap
- Take items out of a heap

Heaps

- Perhaps reduce the number of threads in thread heavy applications.
- You can implement a heap-based priority queue and a timer

The End