Data Structures and Algorithms Cheat Sheet

Arrays &Strings

Stores data elements based on an sequential, most commonly 0 based, index.

Time Complexity

- **Indexing:** Linear array: O(1), Dynamic array: O(1)
- **Search:** Linear array: O(n), Dynamic array: O(n)
- **Optimized Search:** Linear array: O(log n), Dynamic array: O(log n)
- **Insertion:** Linear array: n/a, Dynamic array: O(n)

Bonus:

- type[] name = {val1, val2, ...}
- Arrays.sort(arr) -> O(n log(n))
- Collections.sort(list) -> O(n log(n))
- int digit = '4' - '0' -> 4
- String s = String.valueOf('e') -> "e"
- (int) 'a' -> 97 (ASCII)
- new String(char[] arr) ['a','e'] -> "ae"
- (char) ('a' + 1) -> 'b'
- Character.isLetterOrDigit(char) -> true/false
- new ArrayList<>(anotherList); -> list w/ items
- StringBuilder.append(char||String)

Linked List

Stores data with nodes that point to other nodes.

Time Complexity

- **Indexing:** O(n)
- **Search:** O(n)
- **Optimized Search:** O(n)
- **Append:** O(1)
- **Prepend:** O(1)
- **Insertion:** O(n)

HashTable

Stores data with key-value pairs.

Time Complexity

- **Indexing:** O(1)
- **Search:** O(1)
- **Insertion:** O(1)

Bonus:

- {1, -1, 0, 2, -2} into map

HashMap {-1, 0, 2, 1, -2} -> any order
LinkedHashMap {1, -1, 0, 2, -2} -> insertion order
TreeMap {-2, -1, 0, 1, 2} -> sorted

- Set doesn't allow duplicates.
- map.getOrDefaultValue(key, default value)

DFS vs BFS

DFS	BFS
• Better when target is closer to Source.	• Better when target is far from Source.
• Stack -> LIFO	• Queue -> FIFO
• Preorder, Inorder, Postorder Search	• Level Order Search
• Goes deep	• Goes wide
• Recursive	• Iterative
• Fast	• Slow

Data Structures and Algorithms Cheat Sheet

Binary Search - Iterative

```
public int binarySearch(int target, int[] array) {
   int start = 0;
   int end = array.length - 1;
   while (start <= end) {
      int middle = start + ((end - start) / 2);
      if (target == array[middle]) {
          return target;
      } else if (search < array[middle]) {
          end = middle - 1;
      } else {
          start = middle + 1;
      }
   }
```

Binary Search Big O Notation

	Time	Space
Binary Search	O(log n)	O(1)

Binary Search - Recursive

```
public int binarySearch(int search, int[] array,
int start, int end) {
   int middle = start + ((end - start) / 2);
   if(end < start) {
      return -1;
   }
   if (search == array[middle]) {
      return middle;
   } else if (search < array[middle]) {
      return binarySearch(search, array, start,
middle - 1);
   } else {
      return binarySearch(search, array, middle +
1, end);
   }
}
```

Data Structures and Algorithms Cheat Sheet

Binary Search - Iterative (cont)

```
    return -1;
}
```

Bit Manipulation

Sign Bit	0 -> Positive, 1 -> Negative
AND	0 & 0 -> 0 0 & 1 -> 0 1 & 1 -> 1
OR	0 | 0 -> 0 0 | 1 -> 1 1 | 1 -> 1
XOR	0 ^ 0 -> 0 0 ^ 1 -> 1 1 ^ 1 -> 0
INVERT	~ 0 -> 1 ~ 1 -> 0

Bonus:

- **Shifting**

- Left Shift

0001 << 0010 (Multiply by 2)

- Right Shift

0010 >> 0001 (Division by 2)

- Count 1's of n, Remove last bit n = n & (n-1);
- Extract last bit

n&-n or n&~(n-1) or n^(n&(n-1))

- n ^ n -> 0
- n ^ 0 -> n

Sorting Big O Notation

	Best	Average	Space
Merge Sort	O(n log(n))	O(n log(n))	O(n)
Heap Sort	O(n log(n))	O(n log(n))	O(1)
Quick Sort	O(n log(n))	O(n log(n))	O(log(n))
Insertion Sort	O(n)	O(n^2)	O(1)
Selection Sort	O(n^2)	O(n^2)	O(1)
Bubble Sort	O(n)	O(n^2)	O(1)

Merge Sort

```
private void mergesort(int low, int high) {
if (low < high) {
    int middle = low + (high - low) / 2;
    mergesort(low, middle);
    mergesort(middle + 1, high);
    merge(low, middle, high);
  }
}
private void merge(int low, int middle, int high)
{
for (int i = low; i <= high; i++) {
    helper[i] = numbers[i];
}
int i = low;
int j = middle + 1;
int k = low;
while (i <= middle && j <= high) {
  if (helper[i] <= helper[j]) {
    numbers[k] = helper[i];
    i++;
  } else {
    numbers[k] = helper[j];
    j++;
  }
  k++;
}
while (i <= middle) {
  numbers[k] = helper[i];
  k++;
  i++;
 }
}
```

Quick Sort

Insertion Sort

```
void insertionSort(int arr[]){
  int n = arr.length;
  for (int i = 1; i < n; ++i){
      int key = arr[i];
      int j = i - 1;
      while (j >= 0 && arr[j] > key) {
           arr[j + 1] = arr[j];
           j = j - 1;
      }
      arr[j + 1] = key;
   }
}
```


```
private void quicksort(int low, int high) {
int i = low, j = high;
int pivot = numbers[low + (high-low)/2];
while (i <= j) {
    while (numbers[i] < pivot){
      i++;
    }
    while (numbers[j] > pivot){
      j--;
    }
    if (i <= j) {
      exchange(i, j);
      i++;
      j--;
    }
}
if (low < j)
    quicksort(low, j);
if (i < high)
    quicksort(i, high);
}
```