1 Libraries

1.1 Heap

```
import heapq

a = [1,2,3,4,5,6]
# Heapq defaults to min heap
heapq.heapify(a) # makes a into a min heap object
heapq.heappop(a)

heapq._heapify_max(a) # Makes a into max heap
heapq._heappop_max(a)
```

1.2 Queue

```
import queue

q = queue.Queue()

q.empty()
q.full() # if maxsize specified
q.put(item)
q.get()
q.qsize()
```

1.3 Priority Queue

```
import queue
pq = queue.PriorityQueue()

pq.put((10, 'ten'))
pq.put((1, 'one'))
pq.put((5, 'five'))

x = []
while not pq.empty():
    print(pq.get())
```

2 Common Routines

2.1 Arrays

2.1.1 Binary Search

```
def binary_search(A, target):
    low, high = 0, len(A)-1
    while low <= high:
        mid = (low + high) // 2

        if A[mid] == target:
            return mid
        elif A[mid] < target:
            low = mid + 1
        else:
            high = mid - 1
    return -1
```

2.1.2 Partition

```
def partition(A, left, right, idx):
    value = A[idx]
    new_pivot_idx = left
    A[idx], A[right] = A[right], A[idx]

    for i in range(left, right):
        if comp(A[i], value):
            A[i], A[new_pivot_idx] = A[new_pivot_idx], A[i]
            new_pivot_idx += 1
        A[right], A[new_pivot_idx] = A[new_pivot_idx], A[right]
    return new_pivot_idx
```

2.1.3 Linked Lists

1. Reverse Sub-list

```
def reverse_sublist(L, start, finish):
    dummy_head = sublist_head = ListNode(0, L)
    for _ in range(1, start):
        sublist_head = sublist_head.next

    sublist_iter = sublist_head.next

    for _ in range(finish-start):
        temp = sublist_iter.next
        sublist_head.next, sublist_iter.next, temp.next = \
            temp, temp.next, sublist_head.next

    return dummy_head.next
```

2. Cycle Finding

```
def has_cycle(head):
    fast = slow = head
    while fast and fast.next and fast.next.next:
        slow, fast = slow.next, fast.next.next
        if slow is fast:
            return True
    return False
```

2.1.4 Reverse Linked List

```
def reverse_linked_list(head):
    prev = None
    curr = head
    while curr:
        nxt = curr.next
        curr.next = prev
        prev = curr
        curr = nxt
    return prev
```

2.2 Graph Routines

2.2.1 BFS

```
from queue import Queue

def bfs(node):
    q = Queue()
    q.put(node)
    visited = set()
    visited.add(node)

    while not q.empty():
        n = q.get()
        visit(n)

        for neighbour in n.neighbours:
            if neighbour not in visited:
                q.put(neighbour)
                visited.add(neighbour)
```

2.2.2 DFS

```
def dfs(node):
    stack = [node]
    visited = set()
    visited.add(node)
    while stack:
        n = stack.pop()
        visit(n)
        for neighbour in n.neighbours:
            if neighbour not in visited:
            stack.append(neighbour)
            visited.add(neighbour)
```