**Problem 1**

QuickSort ([1, 6, 2, 4, 3, 5])

E = 1;

L = [];

G = [6, 2, 4, 3, 5];

QuickSort([]);

QuickSort([6, 2, 4, 3, 5]);

E = 6;

L = [2, 4, 3, 5];

G = [];

QuickSort([]);

QuickSort([2, 4, 3, 5]);

E = 2;

L = [];

G = [4, 3, 5];

QuickSort([]);

QuickSort([4, 3, 5]);

E = 4;

L = [3];

G = [5];

QuickSort([3]);

QuickSort([5]);

Return [3, 4, 5];

Return [2, 3, 4, 5];

Return [2, 3, 4, 5, 6];

Return [1, 2, 3, 4, 5, 6];

**Problem 2**

1. Good pivot: 2, 3, 3, 4, 5
2. Yes. 5/9 elements of A are good pivot

**Problem 3**

Pseudo code

int search(array A) {

if A[0] = 0 then return 0

return binarySearch(A, 0, length of A - 1)

}

int binarySearch(array A, int lowerBound, int upperBound) {

if (lowerBound > upperBound) then

return -1

mid 🡨 (lowerBound + upperBound) / 2

if (arr[mid] = mid) then

return mid

if (arr[mid] < mid) then

return binarySearch(A, mid + 1, upperBound) // search RIGHT side

if (arr[mid] > mid) then

return binarySearch(A, lowerBound, mid -1) // search LEFT side

return -1

}

The above algorithm uses binary search to look for value m, which has running time O(logn), in other word o(n)

Please see my **Prob3\_LittleOh.java** to know more detail.

**Problem 4**

* QuickSelect algorithm (with worst case running time O(n)) to select pivots each time.
* Using this algorithm guarantees that all pivots are good pivots:
  + The recursion tree has height O(log n).

=> Running time is O(n log n) in the worst case.

**Problem 5**

S = { 1, 12, 8, 7, -2, -3, 6 }

n/2 = 7/2 = 3

🡪 k = 7 – 3 = 4 then

QuickSort(S {1, 12, 8, 7, -2, -3, 6 }, 4)

P = 1

L = { -2, -3 }

E = { 1 }

G = { 12, 8, 7, 6 }

🡪 k (=4) > |L| + |E| (= 2 + 1 = 3)

QuickSort(G { 12, 8, 7, 6 }, k - |L| - |E| = 1)

P = 12

L = { 8, 7, 6 } 🡪 |L| = 3

E = { 12 } 🡪 |E| = 1

G = { }

🡪 k (= 1) <= |L| (= 3) then

QuickSort(L { 8, 7, 6 }, k = 1)

P = 8

L = { 7, 6 } 🡪 |L| = 2

E = { 8 } 🡪 |E| = 1

G = { }

🡪 k (= 1) <= |L| then

QuickSort(L { 7, 6 }, k = 1)

P = 7

L = { 6 } 🡪 |L| = 1

E = { 7 } 🡪 |E| = 1

🡪 k (= 1) <= |L| (= 1) then

QuickSort(L { 6 }, k = 1)

P = 6

L = { } 🡪 |L| = 0

E = { 6 } 🡪 |E| = 1

🡪 |L| (= 0) < k (= 1) <= |L| + |E| (= 0 + 1 = 1) then

**Return 6**