Selection Sort :-

0 1 2 3 4 5 ← indices

input :- 3 8 5 7 6 9 ← elements

3 [8 5 7 6 9]

i↓ j↓

3 5 | 8 7 6 9

sorted.

3 5 6 | 7 8 9

1st dip

0 1 2 3 4

5 4 3 2 1

1 4 3 2 5

1 2 3 4 5

123 45

1 2 3 4 5. n

lo = hi

lo ... hi-1

1) a function

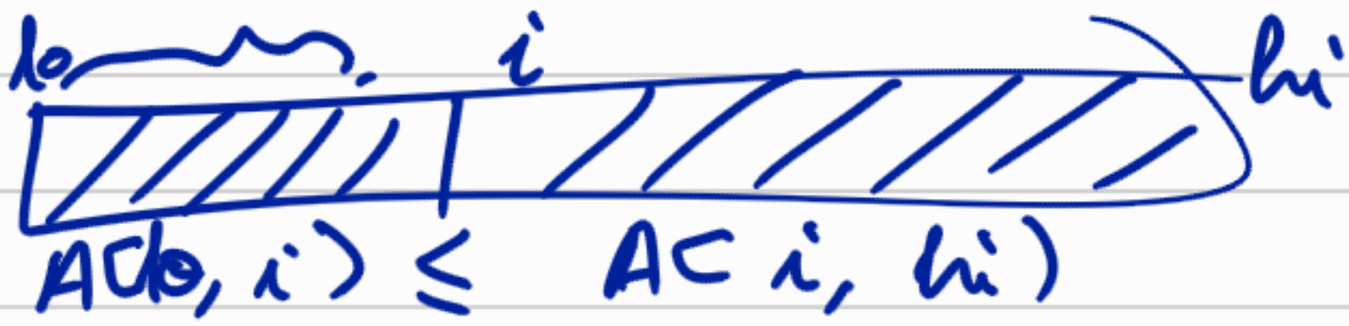
```
int find_min (int[] A, int lo, int hi)
//@requires 0 ≤ lo && lo < hi && hi ≤ len(A);
//@ensures lo ≤ \result && \result < hi;
//@ensures A[\result] ≤ A[lo, hi);
```

2) a function for swapping entries by index:

```
void swap(int[] A, int i, int j);
// @requires 0 <= i, j < \length(A);
// swap exchanges A[i], A[j] leaving
   other entries unchanged.
```

Exercise:- Code the functions above

```
Void selectionsort(int[] A, int lo, int hi)
//@requires 0 ≤ lo ≤ hi ≤ \length(A);
//@ ensures is_sorted(A, lo, hi);
§
   for (int i = lo; i < hi; i++)
   //@loop_invariant lo <= i && i <= hi;
   § int minindex = find_min(A, i, hi);
     swap(A, i, minindex);
   }
}
```



{

```
for ( int i = lo; i < hi ; i++)
//@ loop_invariants  lo <= i && i <= hi ;
//@ loop_invariant  is_sorted (A, lo, i) ;
// loop_invariant comment:  A[lo, i) ≤ A[i, hi) .

    { int minindex = find_min(A, i, hi);
      swap ( A, i, minindex ) ;
    }
}
```

Running cost :- selectionSort (A, 0, n)

repeat the following for n times

find_min (A, i, hi)

swap. — 2

(n - i) ← n.

$i = 0 \qquad i < n.$

$$\sum_{i=0}^{n-1} (n-i) \qquad (n-i)$$

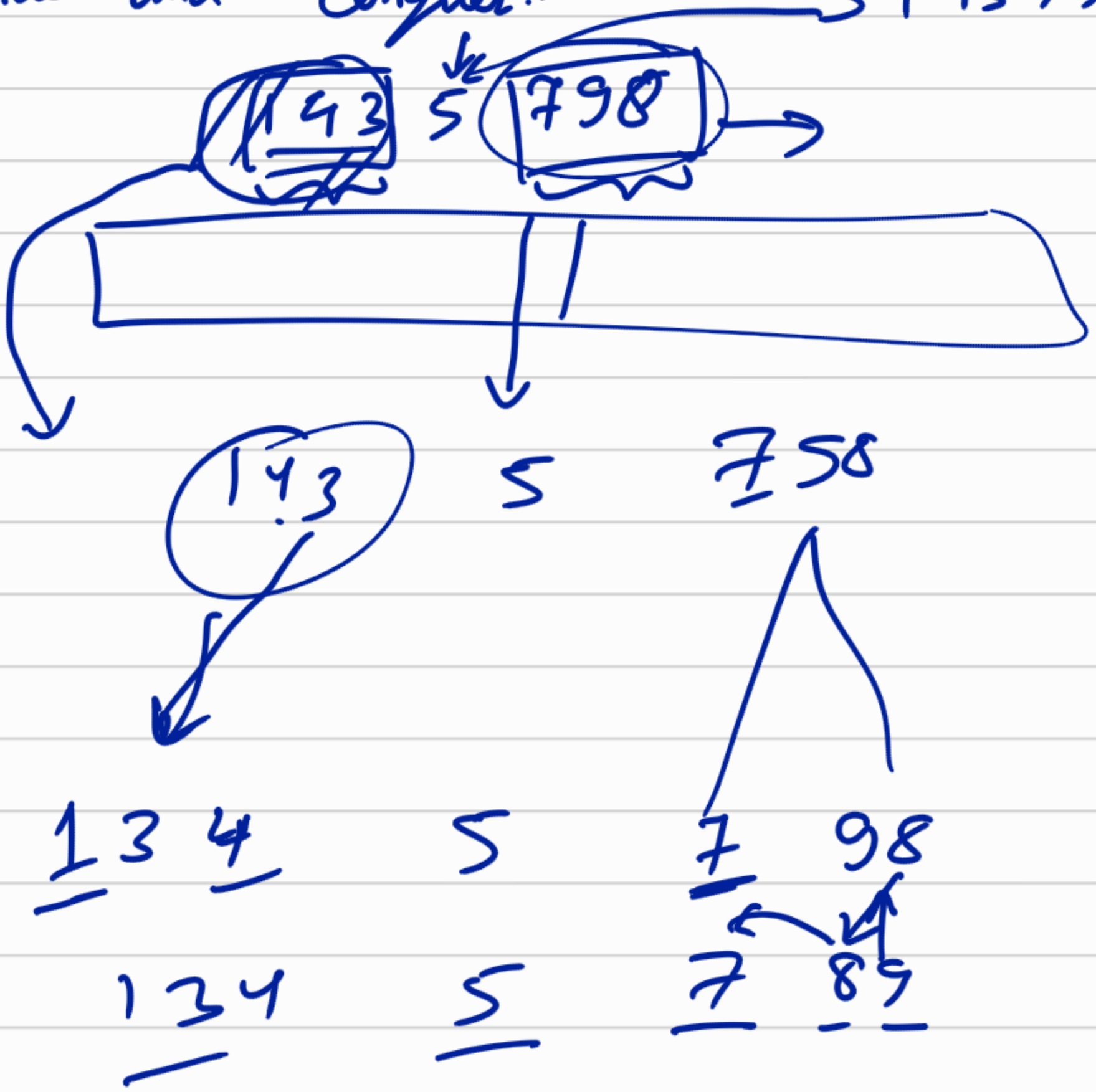
$$n + n-1 , \ldots + 1 .$$

$$= \frac{n(n+1)}{2} + 2n.$$

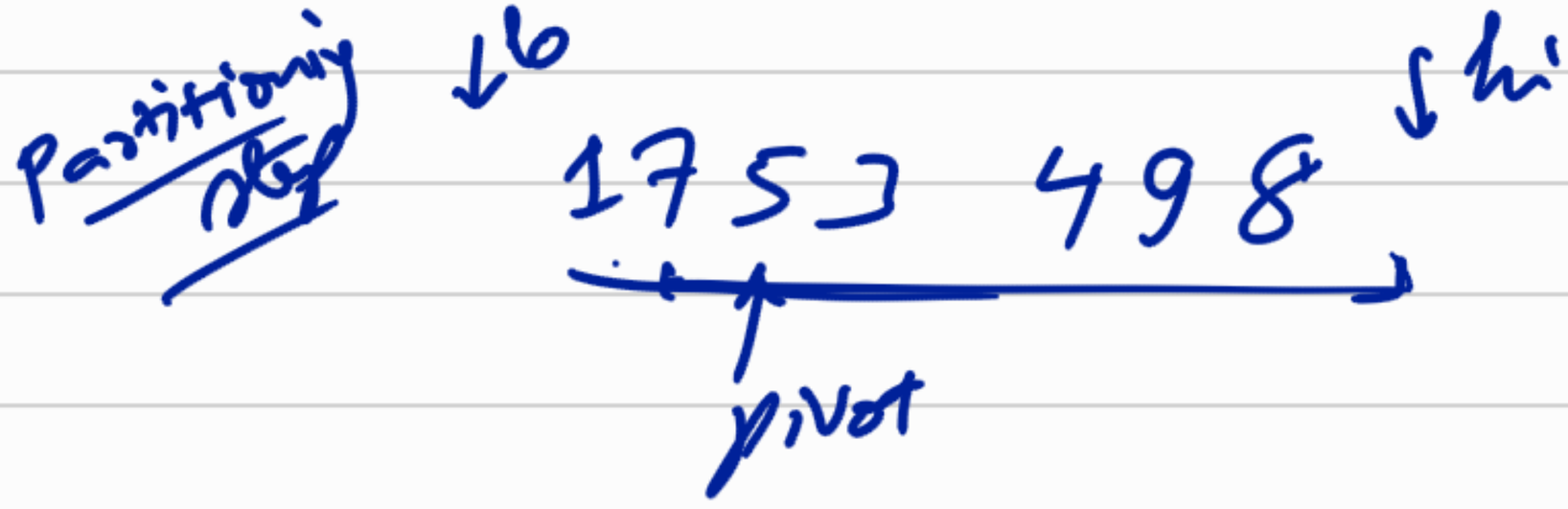
$$\leq$$

$$\in O(n^2).$$

Divide and Conquer :-



Partitioning



A[lo, hi)
pivot

```
int partition( int[] A, int lo, int pivot, int hi)
//@ requires  0 <= lo && lo <= pivot
//@ requires  pivot < hi && hi <= \byte(A);
//@ ensures   lo <= \result < hi;
// must have        A[lo, \result) <= A[\result]
// must have        A[\result] <= A[\result+1, hi).
```

```
void qsort(int[] A, int lo, int hi)
//@ requires 0 <= lo <= hi <= \length(A);
//@ ensures is_sorted(A, lo, hi);
{
  if (hi - lo <= 1) return;
  int pivot = lo + (hi-lo)/2;
  // we really want a random pivot

  int mid = partition(A, lo, pivot, hi);
  qsort(A, lo, mid);
  qsort(A, mid+1, hi);
  return;
}
```

Partition step:

lo pivot hi
1 7 5 3 4 9

lo lo+1 hi
5 7 1 3 4 9

initially
left = lo+1
right = hi

left

right

```
if ( A[lo] < A[left] )
    swap(A, left, right-1);
    right--;
```

lo=0 1 2 3 4 5 hi=6
5 9 1 3 4 7

left=1

right=5

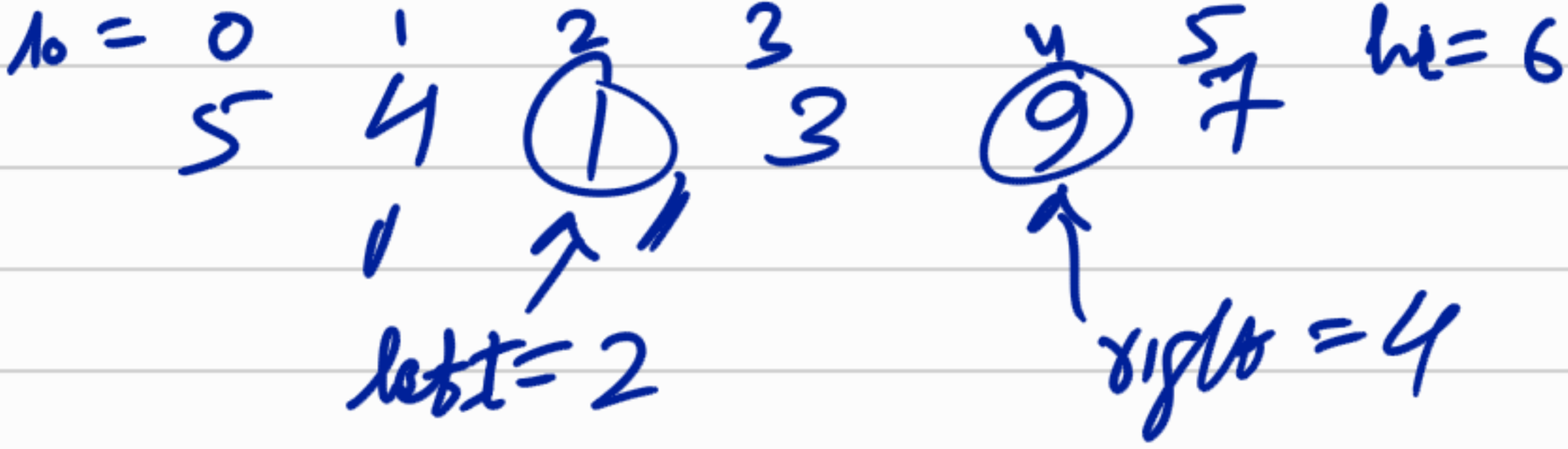
if 5 < 9
swap(A, 1, 5)
right--;

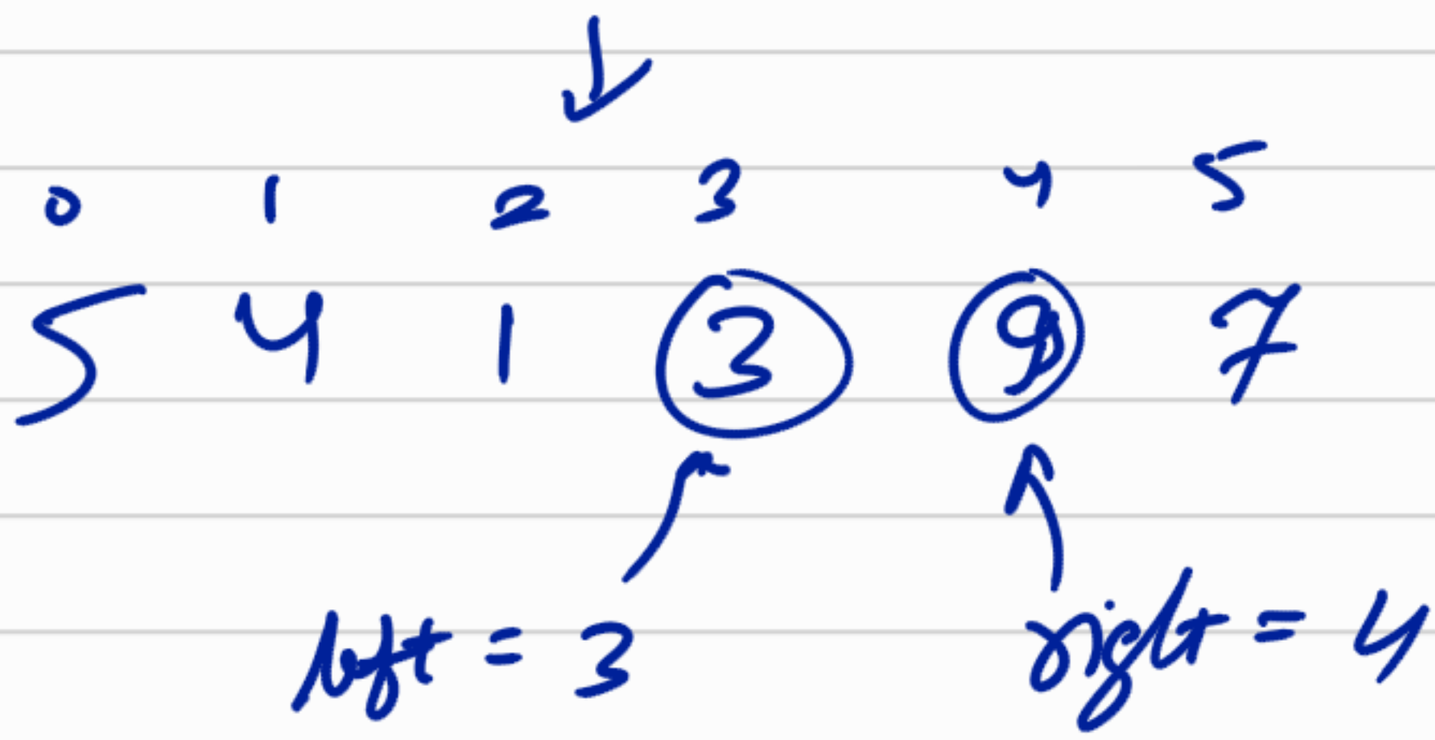
lo=0 1 2 3 4 5 hi=6
5 4 1 3 9 7

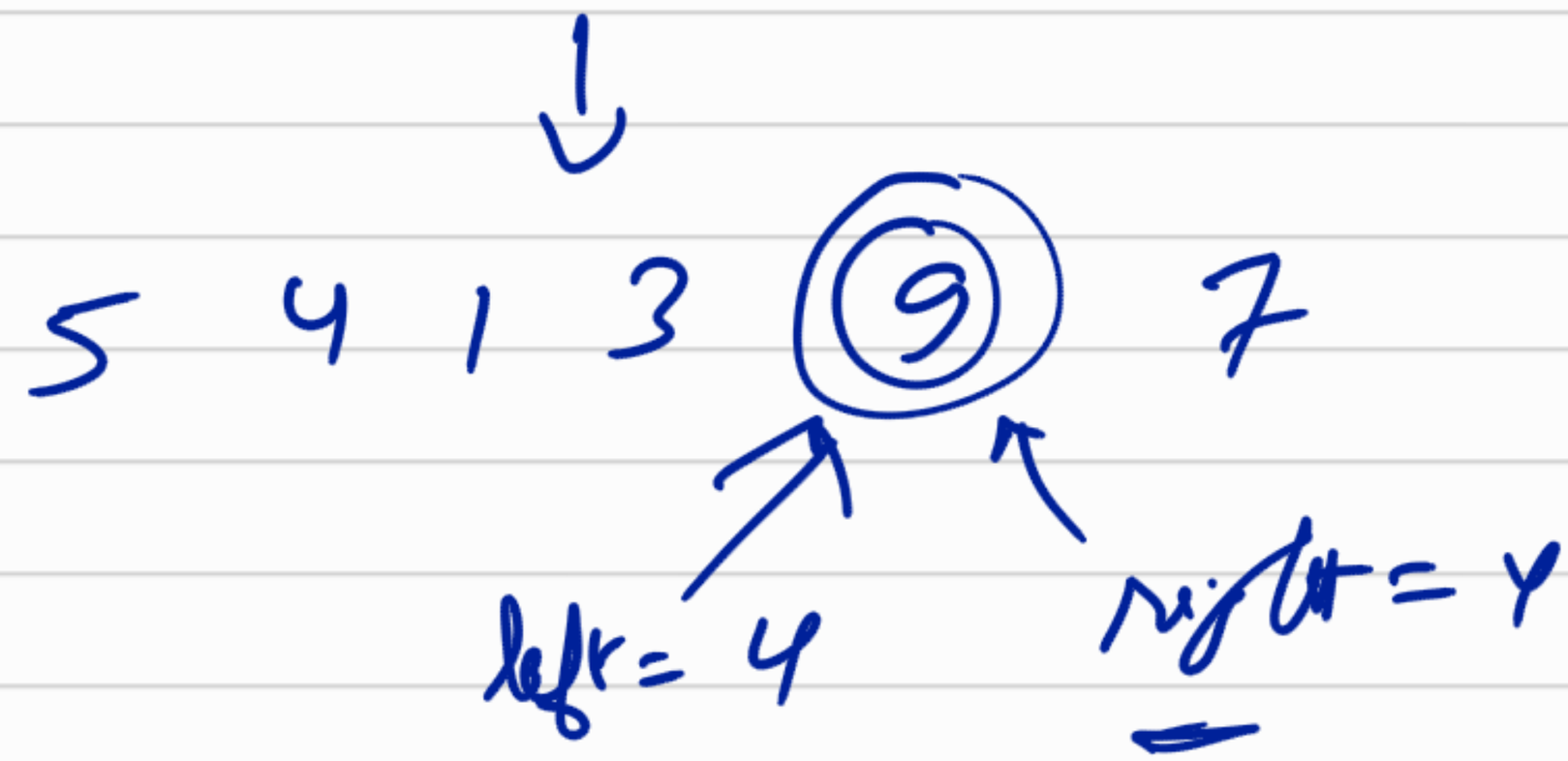
left=1

right=4

if (5 < 4)? 4 ≤ 5







$A[1,4) \leq 5$

$5 < A[4,6)$.

```
                    = pivot
if ( A[lo] < A[left] ) {
        swap( A, left, right-1);
        right --;
}
else
{ //@assert A[left] <= pivot
        left++;
}
```

Loop invariants :-

pivot = A[lo].

(1) pivot < A[right, hi)

(2) pivot >= A[lo+1, left)

(3) lo+1 <= left && left <= right
&& right <= hi

```
int partition (int[] A, int lo, int pi,
               int hi)
//@requires 0<=lo && lo<=pi && pi<hi && hi<=\length(A);
//@ensures lo<= \result && \result<hi;
// ensure that A[\result] < A[\result+1, hi);
//ensure that A[\result] >= A[lo, \result);
```

```
int pivot = A[pi];
swap(A, pi, lo);
int left = lo+1;
int right = hi;

while (left < right)
// LI: lo+1 ≤ left && left ≤ right && right ≤ hi;
// LI2: pivot < A[right, hi)
// LI3: pivot >= A[lo+1, left)
{
        = pivot
  if ( A[lo] < A[left] ){
        swap(A, left, right-1);
        right--;
  }
  else
    { // @assert A[left] <= pivot
        left++;
    }
}
```

```
    //@ assert left == right;
      swap(A, 10, left - 1);
      return left - 1;
}
```

Runtime Complexity of quicksort??

```
Void qsort(int[] A, int lo, int hi)
//@ requires 0 <= lo <= hi <= \length(A);
//@ ensures is_sorted(A, lo, hi);
{
```

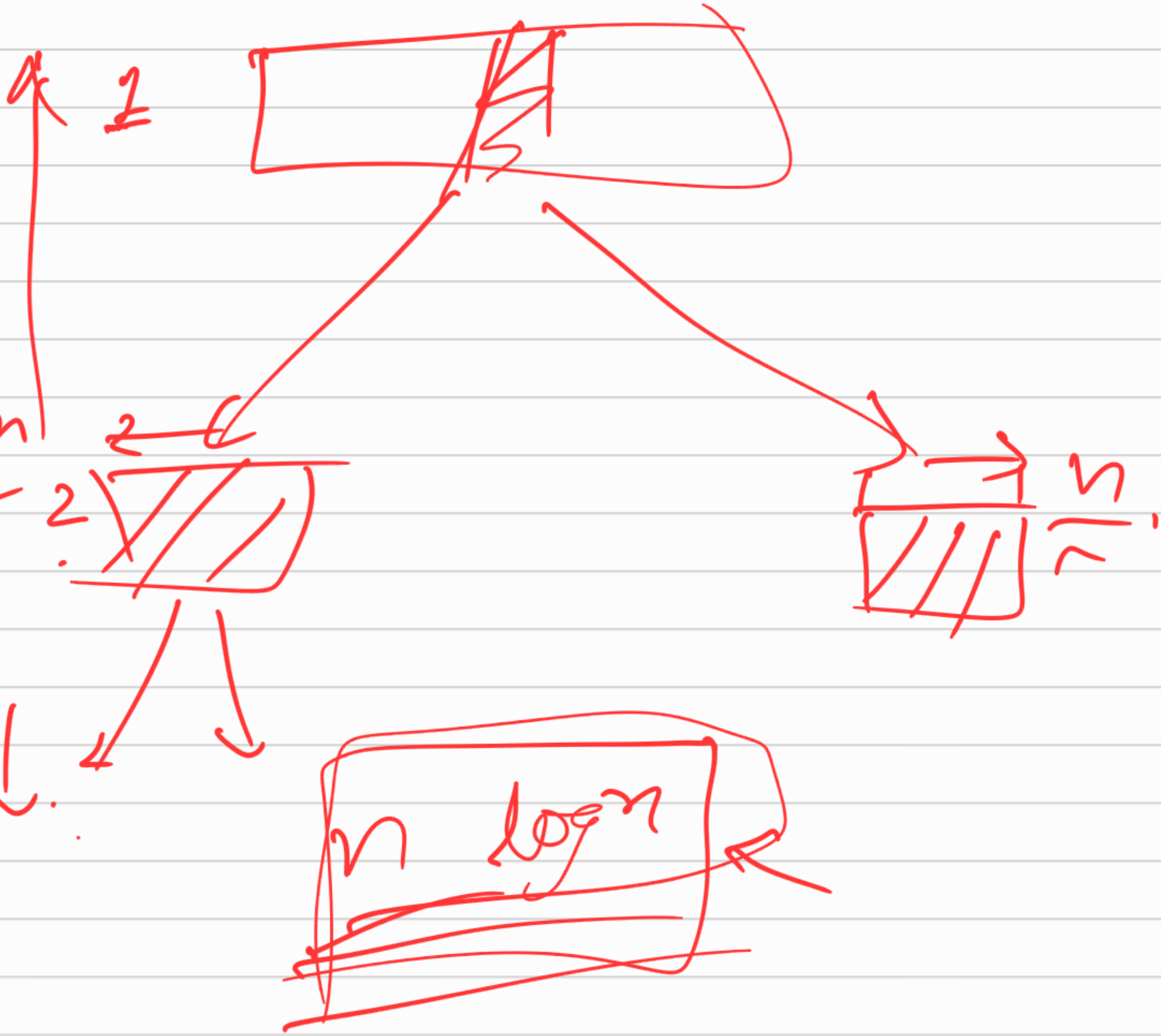
Base case.

```
  if (hi - lo <= 1) return;
  int pivot = lo + (hi-lo)/2;
  // we really want a random pivot
  int mid = partition(A, lo, pivot, hi);
  qsort(A, lo, mid);      ← n/2
  qsort(A, mid+1, hi);    ← n/2.
  return;
}
```

O(n).

Say qsort takes $\phi(n)$ no. of steps

$$\phi(n) \leq c_1 + cn + 2\phi\left(\frac{n}{2}\right).$$


1
log n
2
2
n
n log n