TOURNAMENT SORT

Ralph Eduard So
Jon Francis Pasana
Jameson Adriel Perez
Asherrie Jaye Tan

DEVELOPMENT

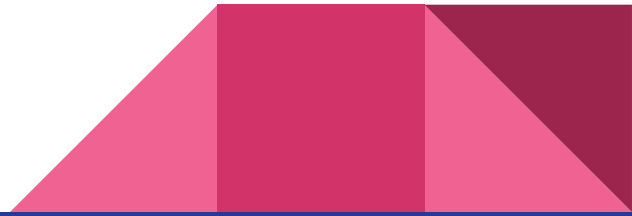
- Alexander Stepanov, Russian computer programmer, who is known best for his primary design and implementation of the c++ library.

- Aaron Kershenbaum, Engineer, who is known best for his book titled “Telecommunications Network Design Algorithm.”

- It was developed in the year 2002.

ORIGINAL VERSION

- It is a generalization of binomial trees of Brown and Vuillemin.
- Selection sort but it uses a priority queue.
- It is a variation of a heapsort and sometimes heapsort can be called tournament sort.
- The name is tournament sort because its similar to a single elimination tournament.



COMPLEXITY

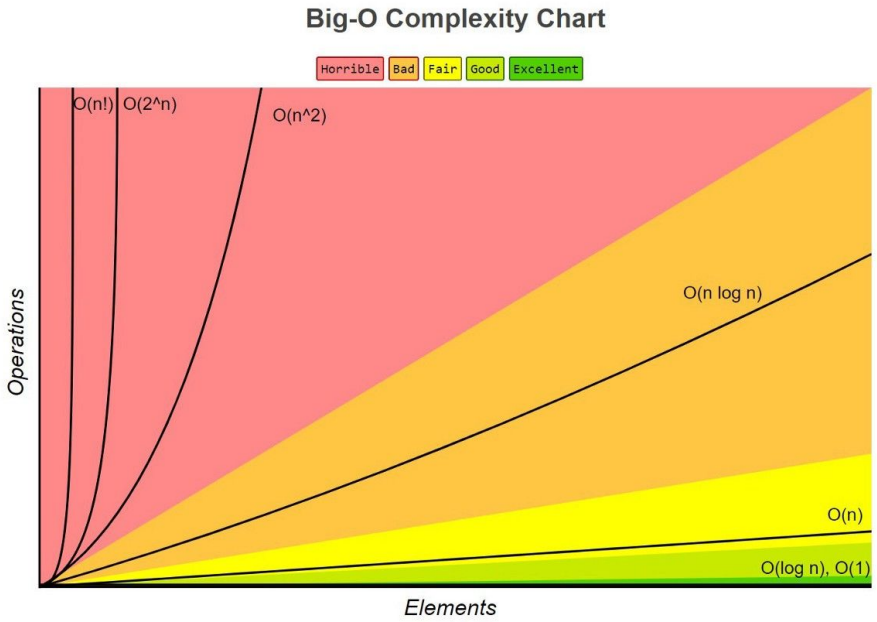
Time: O (n log n)

Space: O(n)

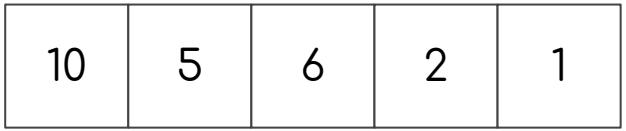
Worst complexity: n*log(n)

Average complexity: n*log(n)

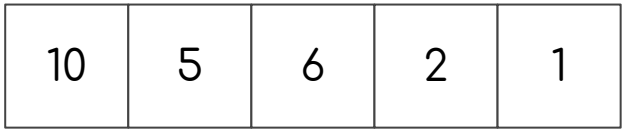
Best complexity: n*log(n)



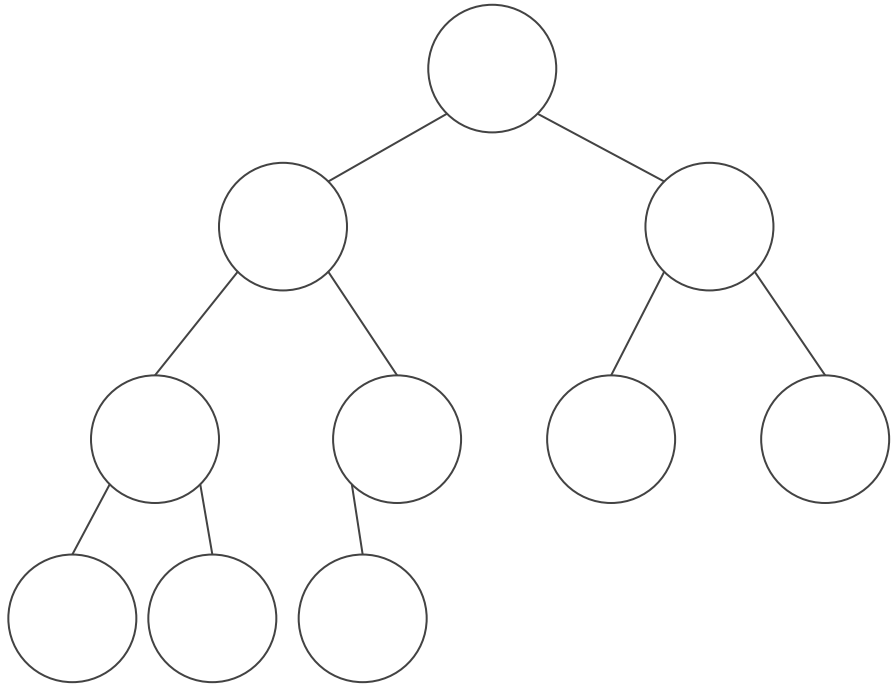
SIMULATION



SIMULATION



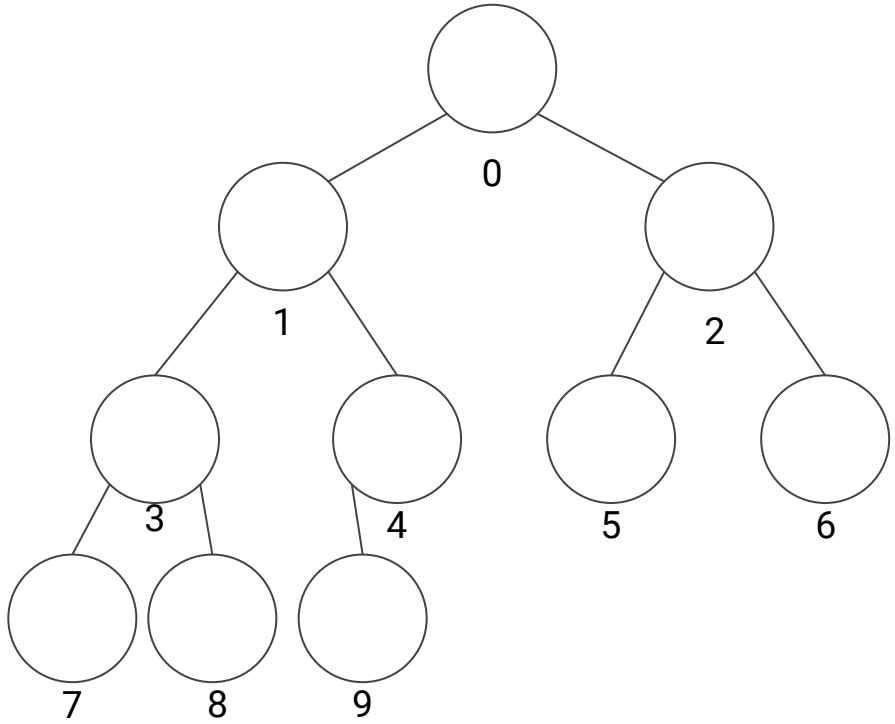
SIMULATION



10	5	6	2	1

					10	5	6	2	1

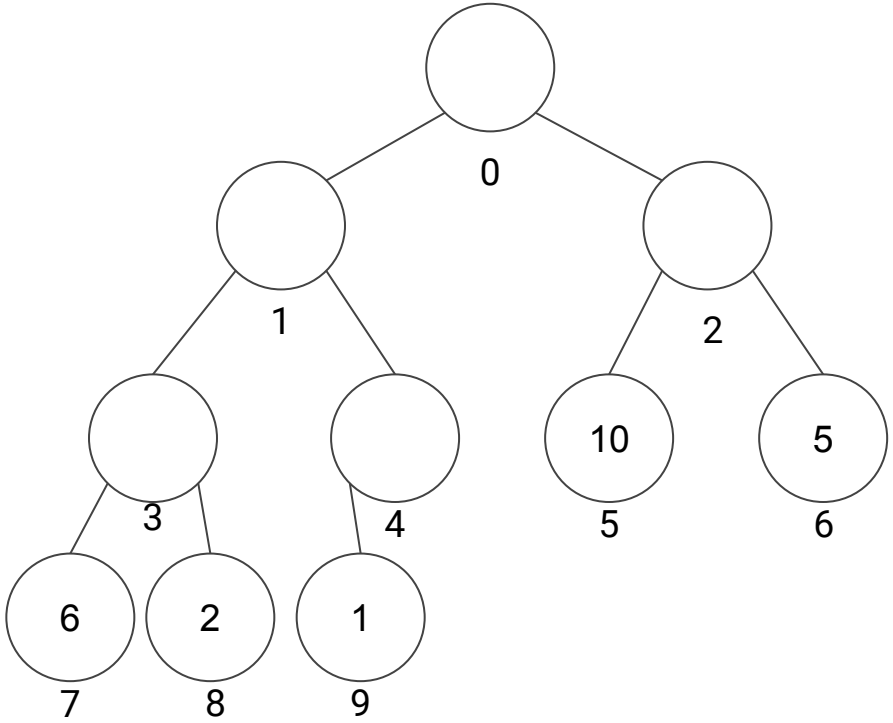
SIMULATION



10	5	6	2	1

					10	5	6	2	1
0	1	2	3	4	5	6	7	8	9

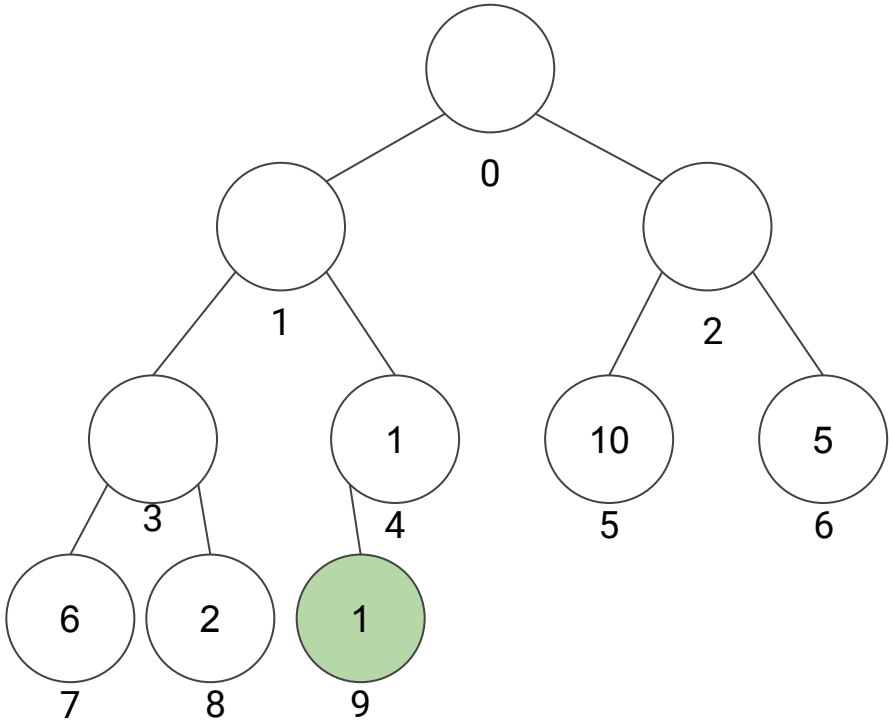
SIMULATION



10	5	6	2	1

					10	5	6	2	1
0	1	2	3	4	5	6	7	8	9

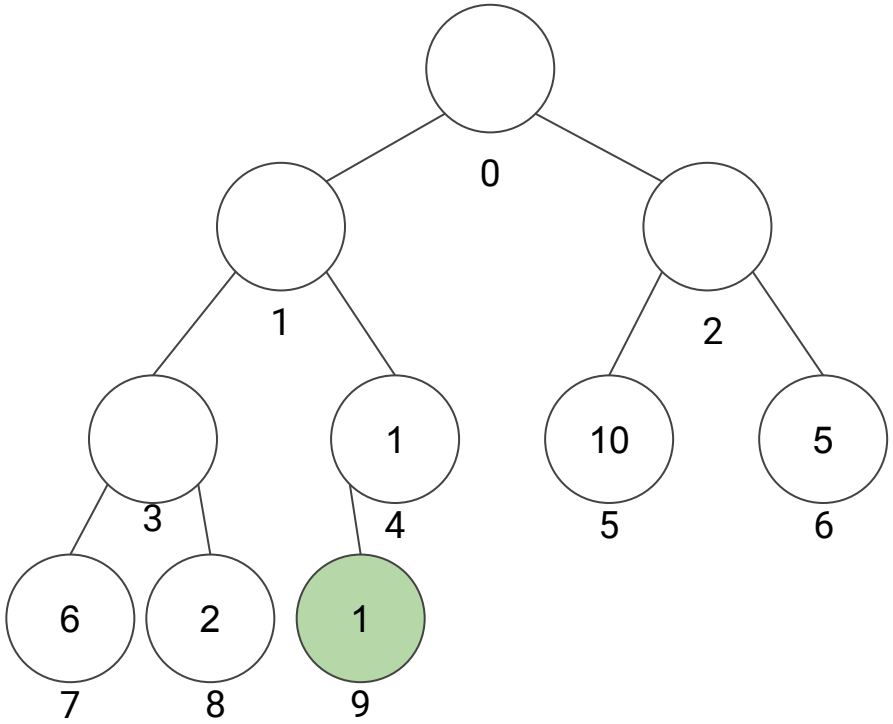
SIMULATION



10	5	6	2	1

					10	5	6	2	1
0	1	2	3	4	5	6	7	8	9

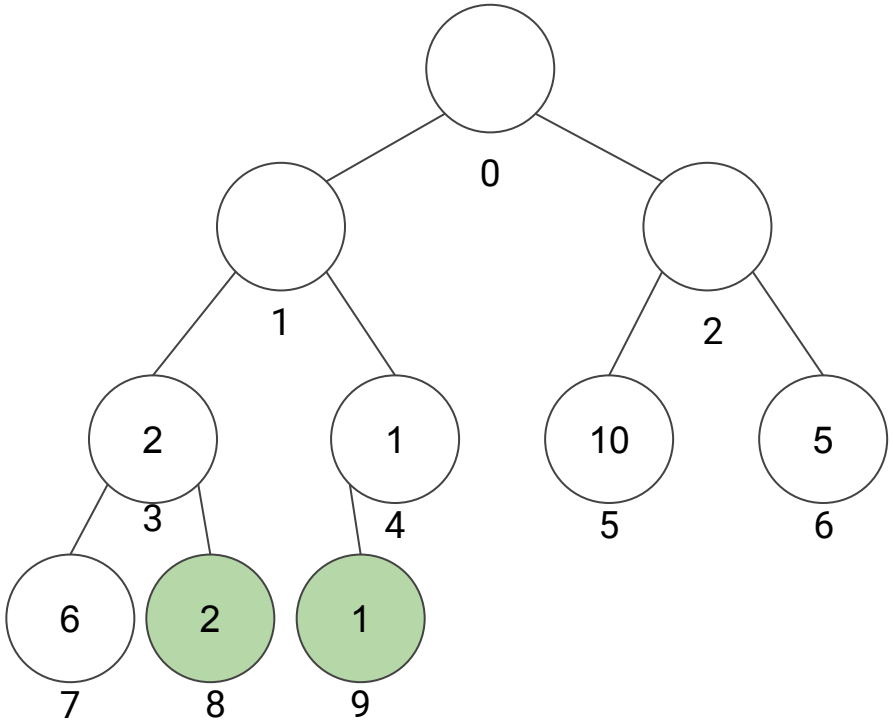
SIMULATION



10	5	6	2	1

				(9)	10	5	6	2	1
0	1	2	3	4	5	6	7	8	9

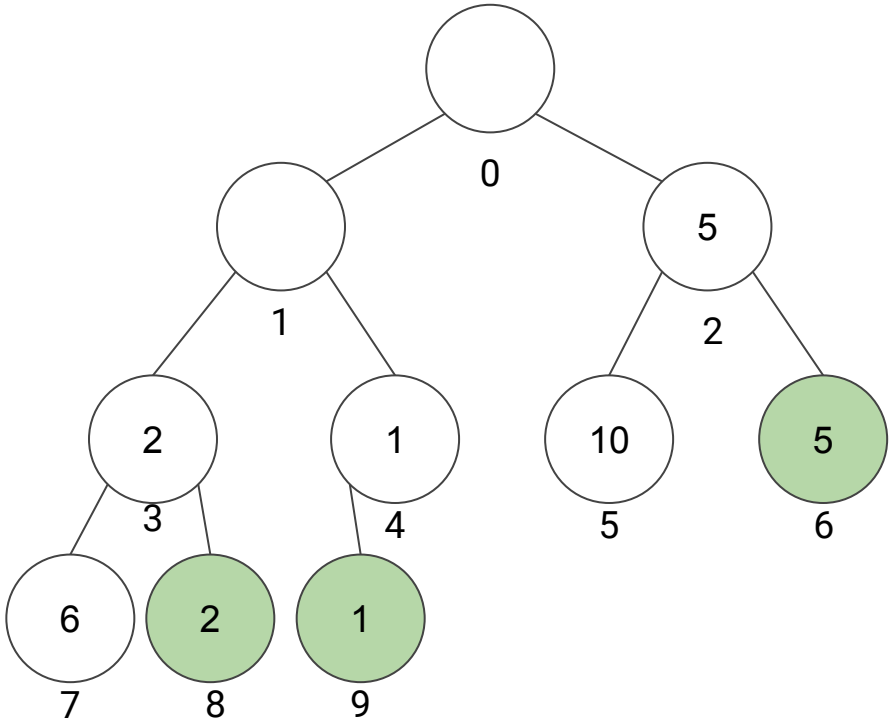
SIMULATION



10	5	6	2	1

			(8)	(9)	10	5	6	2	1
0	1	2	3	4	5	6	7	8	9

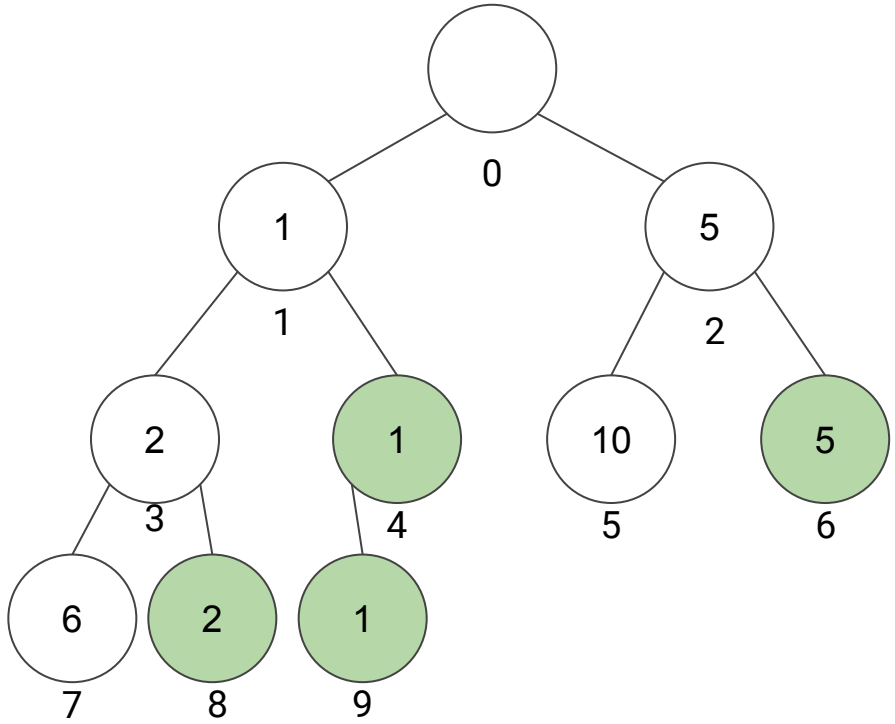
SIMULATION



10	5	6	2	1

		(6)	(8)	(9)	10	5	6	2	1
0	1	2	3	4	5	6	7	8	9

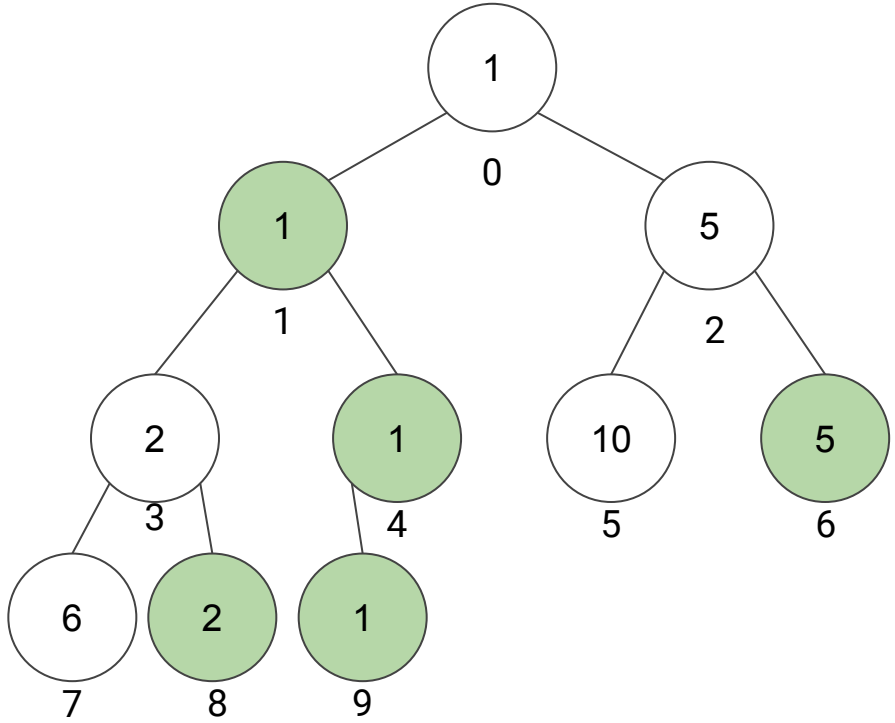
SIMULATION



10	5	6	2	1

	(9)	(6)	(8)	(9)	10	5	6	2	1
0	1	2	3	4	5	6	7	8	9

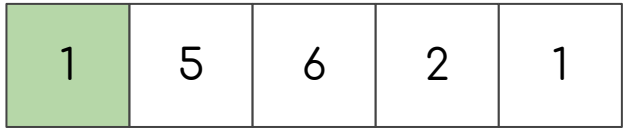
SIMULATION

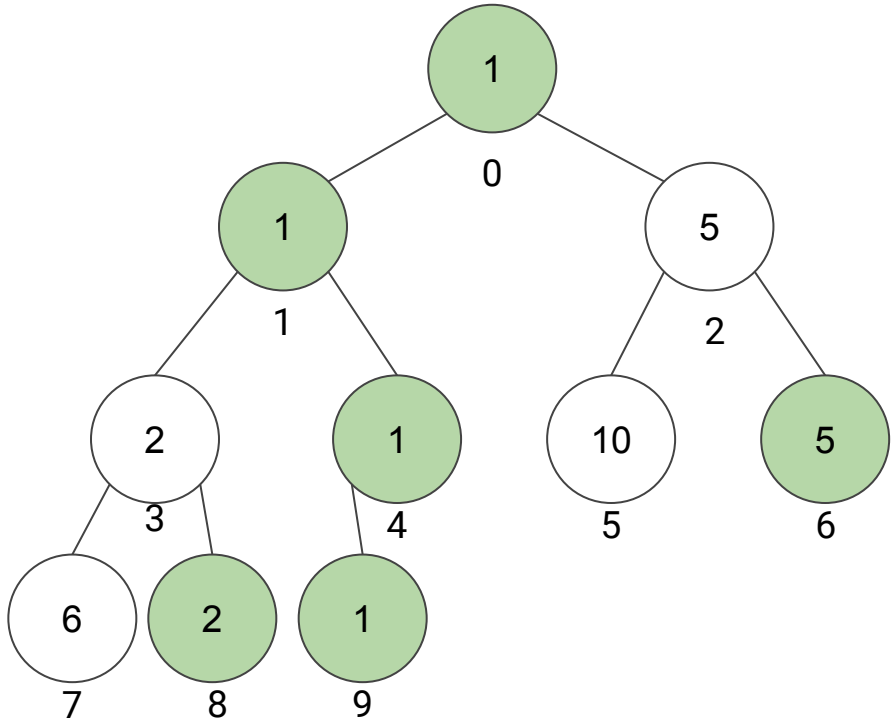


10	5	6	2	1

(9)	(9)	(6)	(8)	(9)	10	5	6	2	1
0	1	2	3	4	5	6	7	8	9

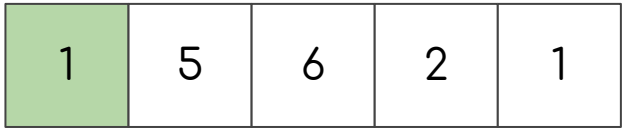
SIMULATION

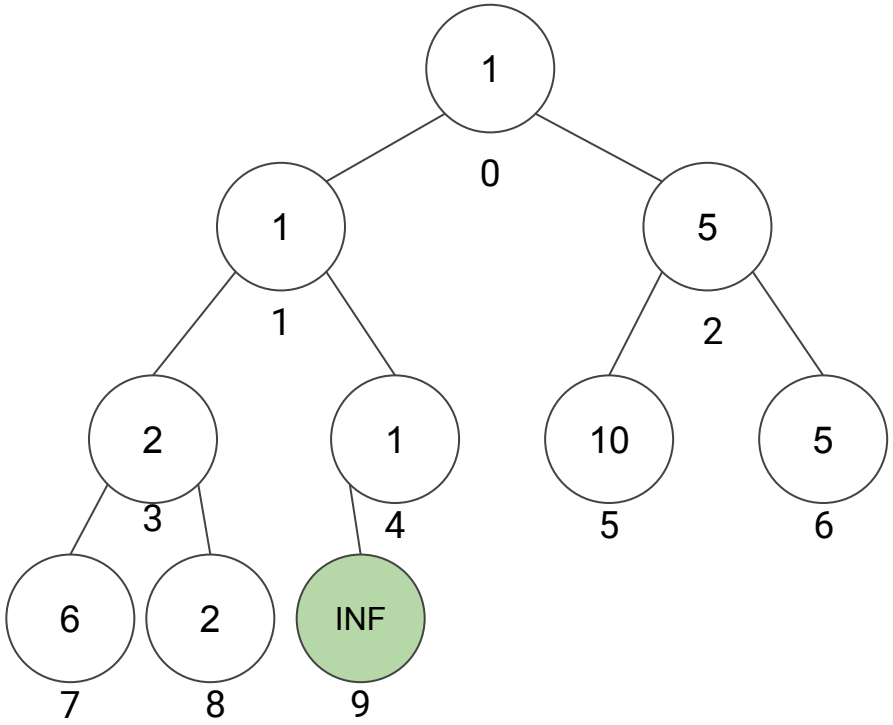




(9)	(9)	(6)	(8)	(9)	10	5	6	2	1
0	1	2	3	4	5	6	7	8	9

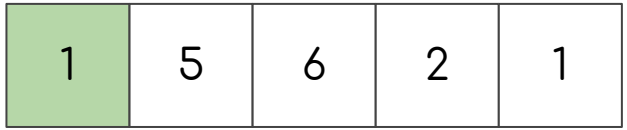
SIMULATION

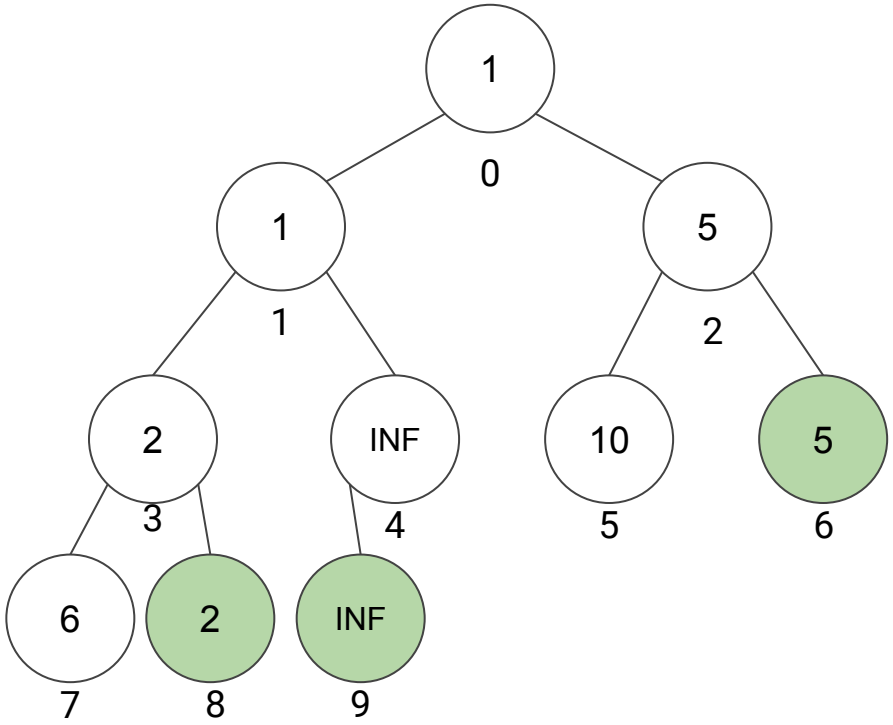




(9)	(9)	(6)	(8)	(9)	10	5	6	2	INF
0	1	2	3	4	5	6	7	8	9

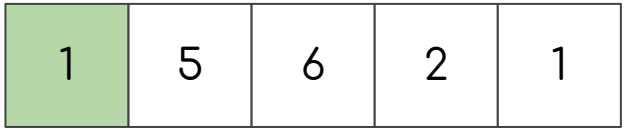
SIMULATION

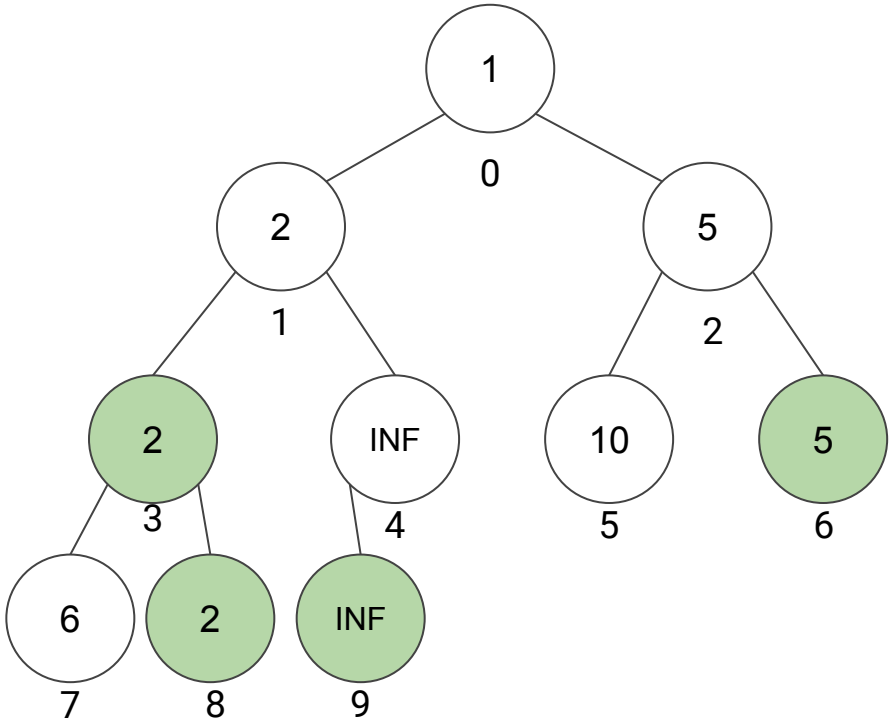




(9)	(9)	(6)	(8)	(9)	10	5	6	2	INF
0	1	2	3	4	5	6	7	8	9

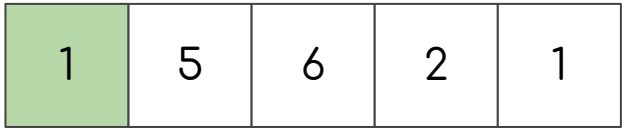
SIMULATION

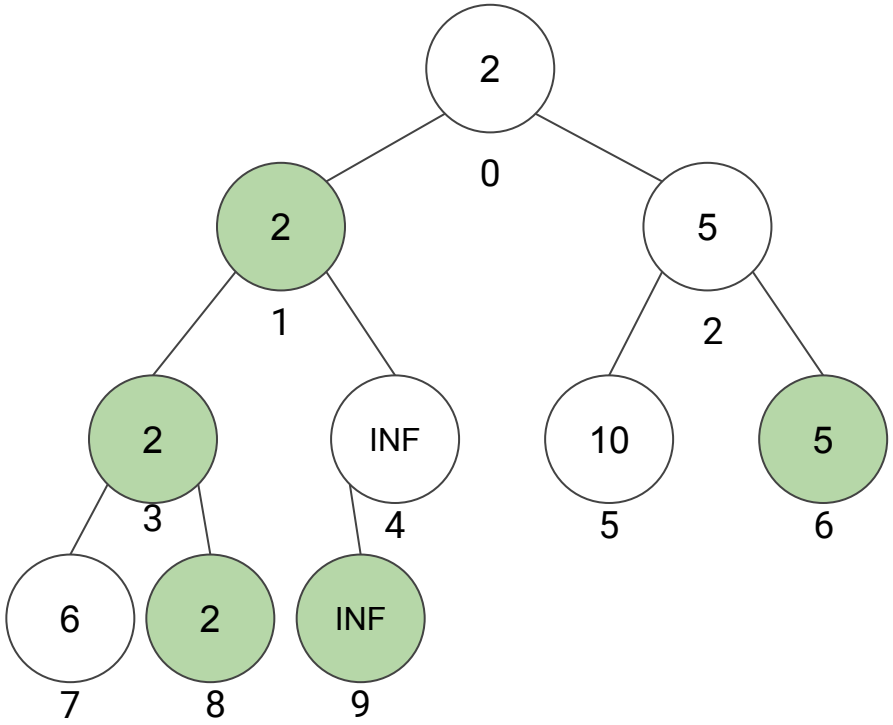




(9)	(8)	(6)	(8)	(9)	10	5	6	2	INF
0	1	2	3	4	5	6	7	8	9

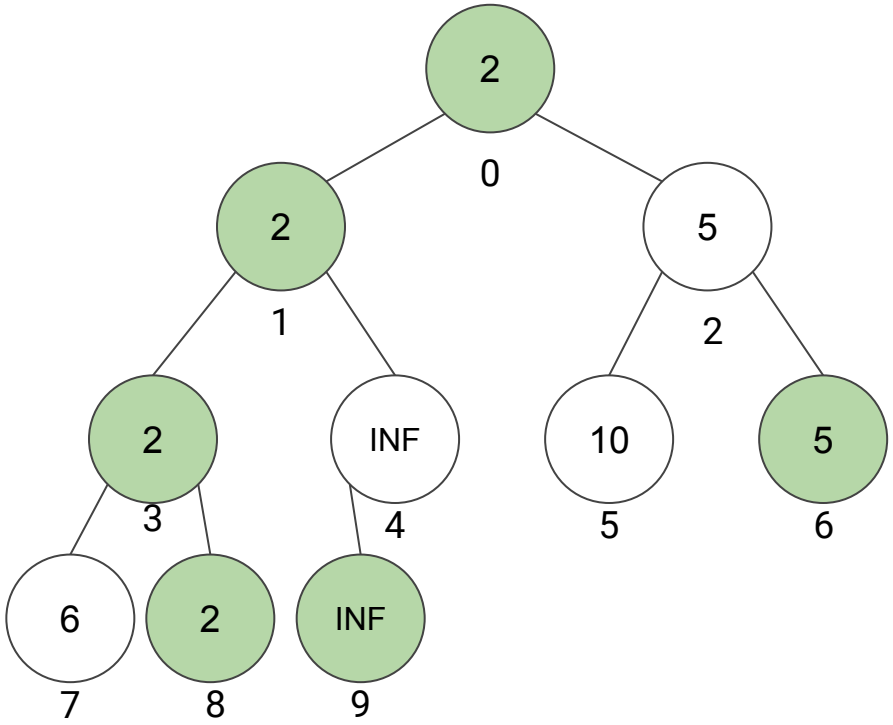
SIMULATION





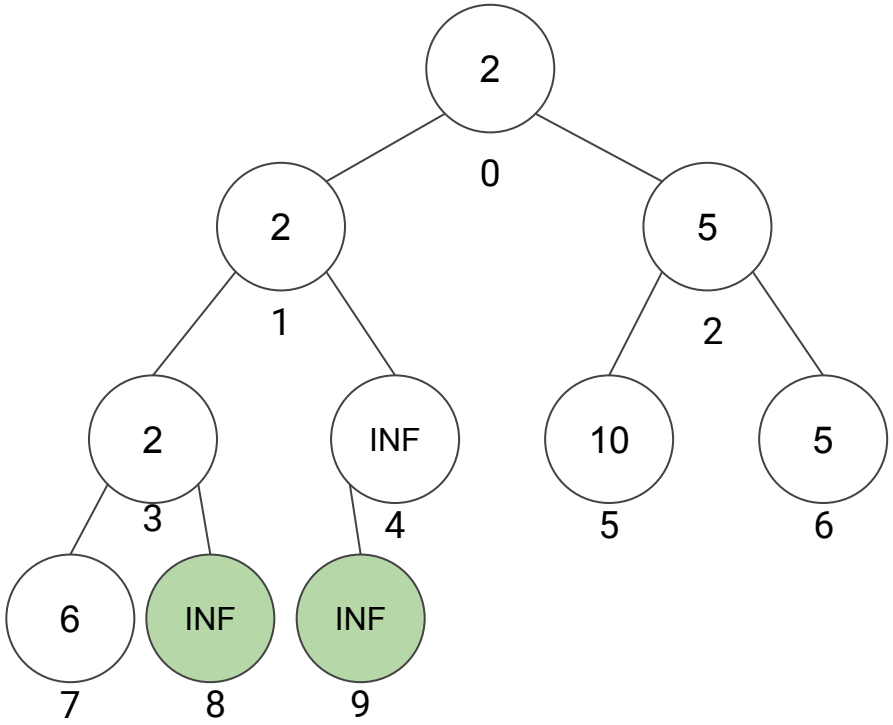
(8)	(8)	(6)	(8)	(9)	10	5	6	2	INF
0	1	2	3	4	5	6	7	8	9

SIMULATION



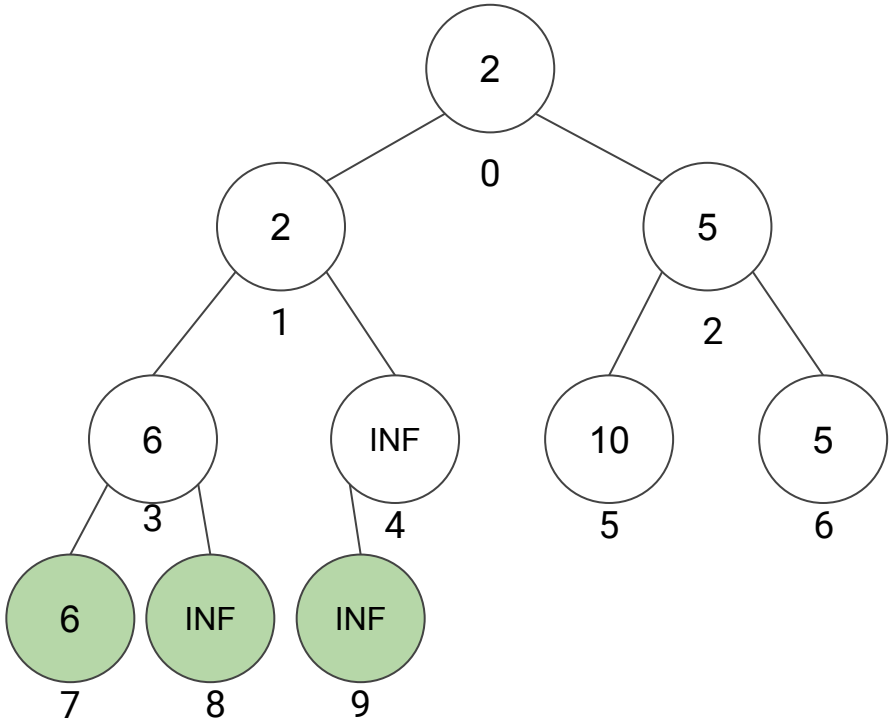
(8)	(8)	(6)	(8)	(9)	10	5	6	2	INF
0	1	2	3	4	5	6	7	8	9

SIMULATION



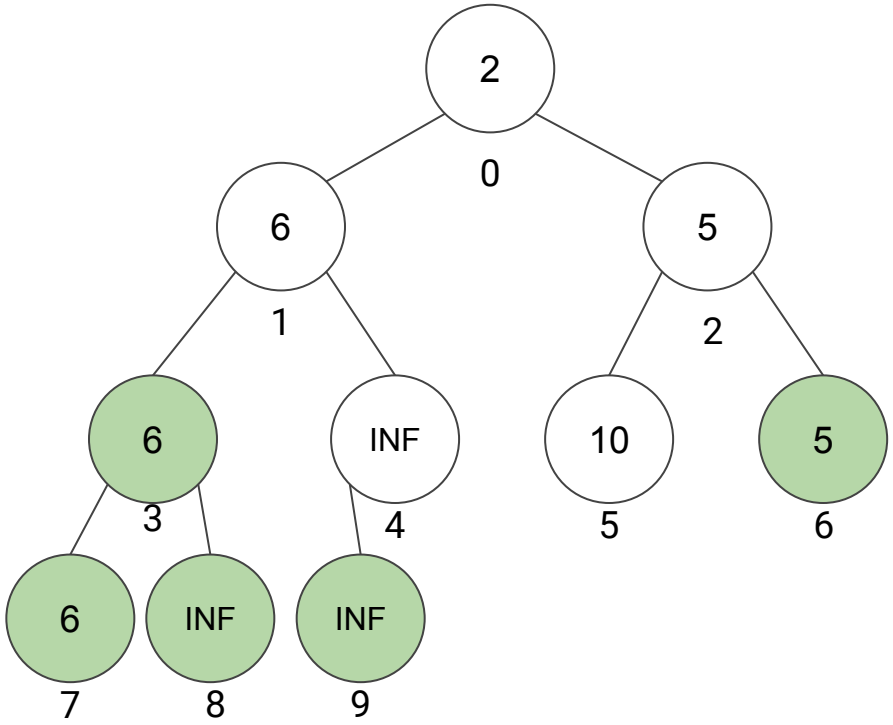
(8)	(8)	(6)	(8)	(9)	10	5	6	INF	INF
0	1	2	3	4	5	6	7	8	9

SIMULATION



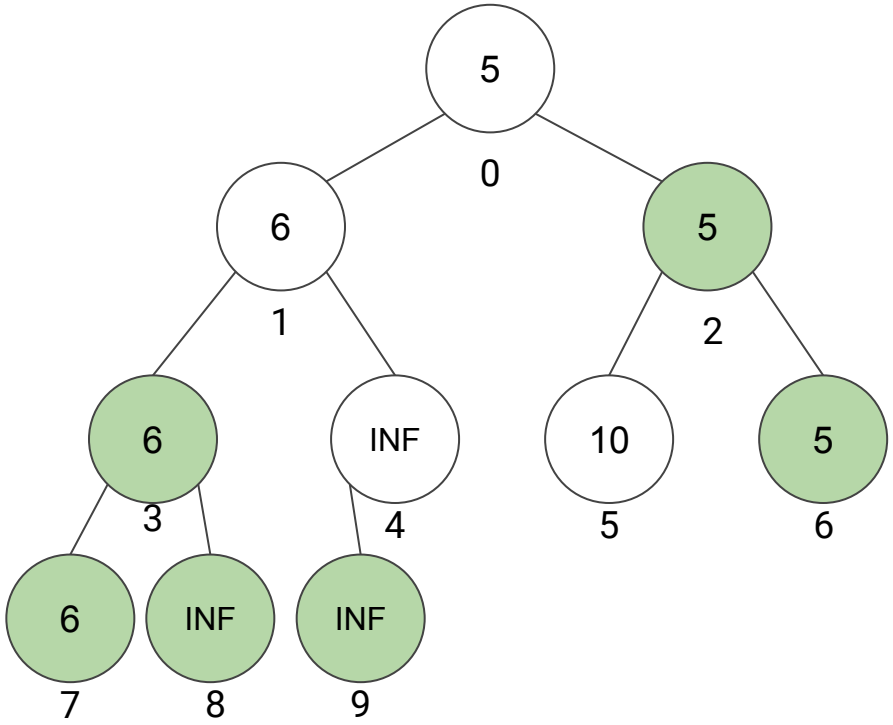
(8)	(8)	(6)	(7)	(9)	10	5	6	INF	INF
0	1	2	3	4	5	6	7	8	9

SIMULATION



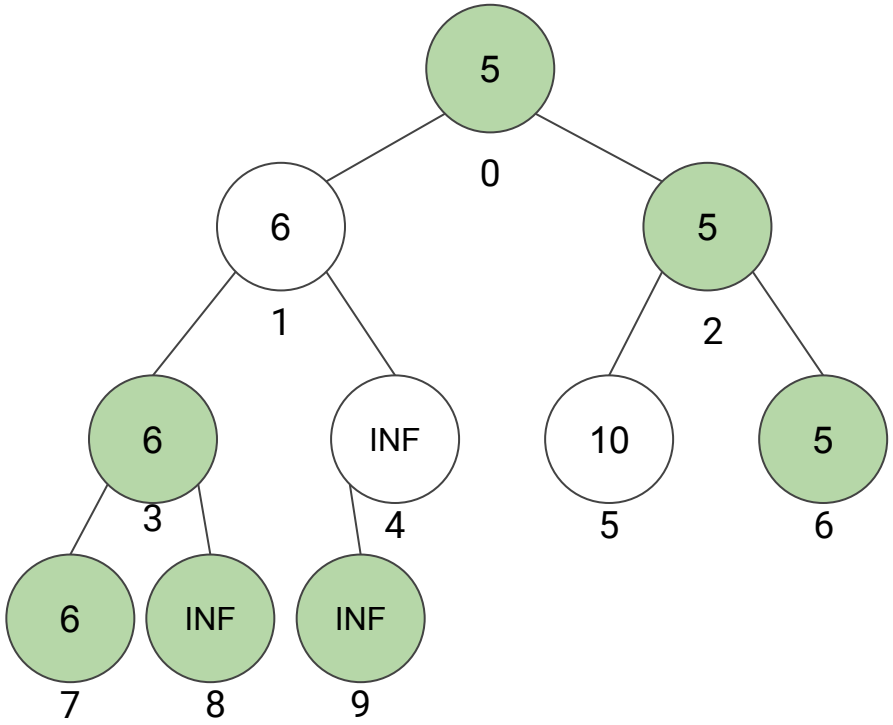
(8)	(7)	(6)	(7)	(9)	10	5	6	INF	INF
0	1	2	3	4	5	6	7	8	9

SIMULATION



(6)	(7)	(6)	(7)	(9)	10	5	6	INF	INF
0	1	2	3	4	5	6	7	8	9

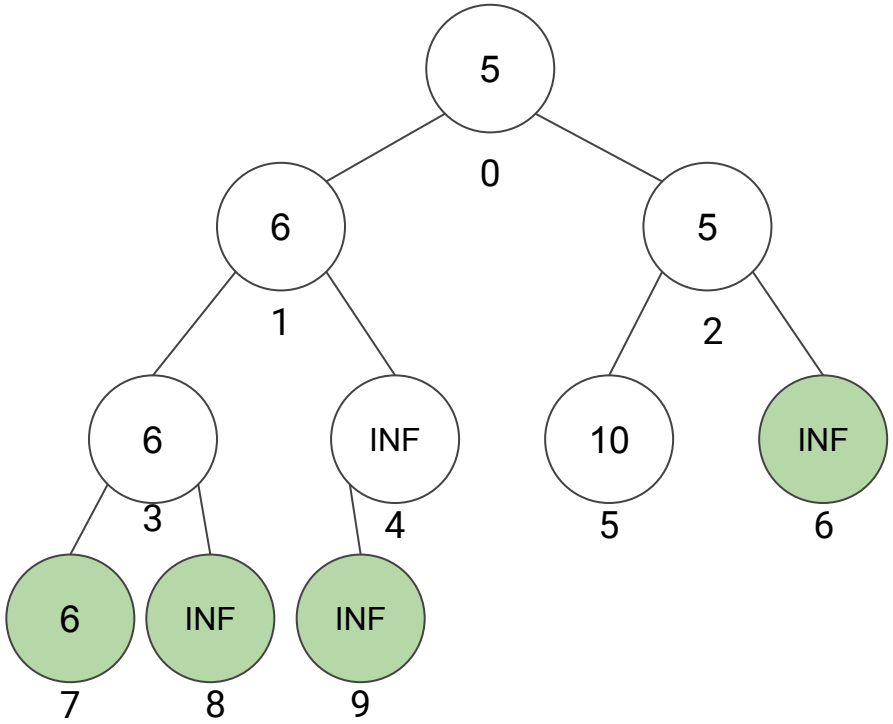
SIMULATION



(6)	(7)	(6)	(7)	(9)	10	5	6	INF	INF
0	1	2	3	4	5	6	7	8	9

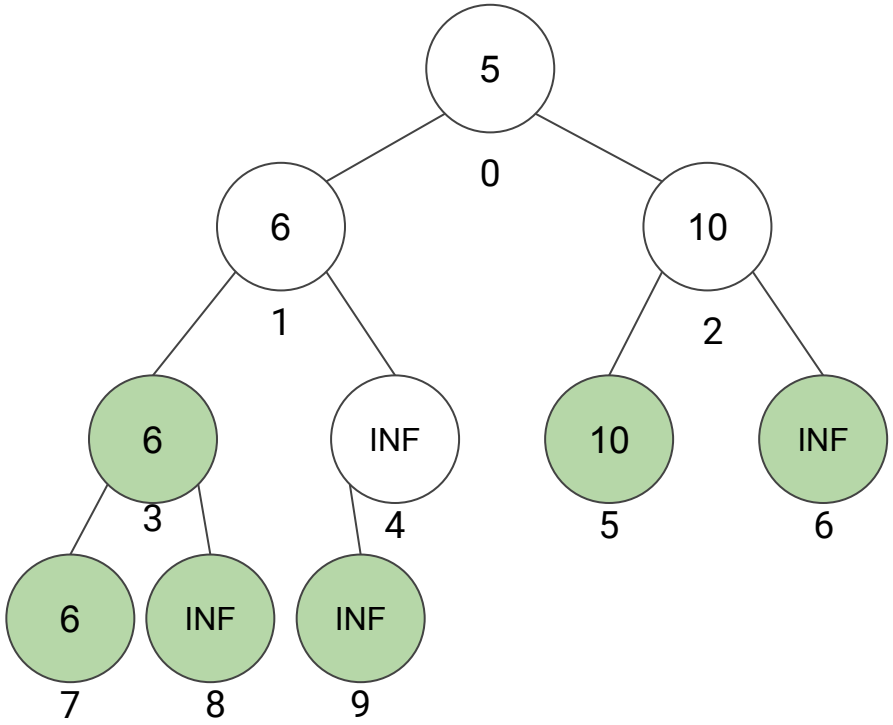
SIMULATION

1	2	5	2	1



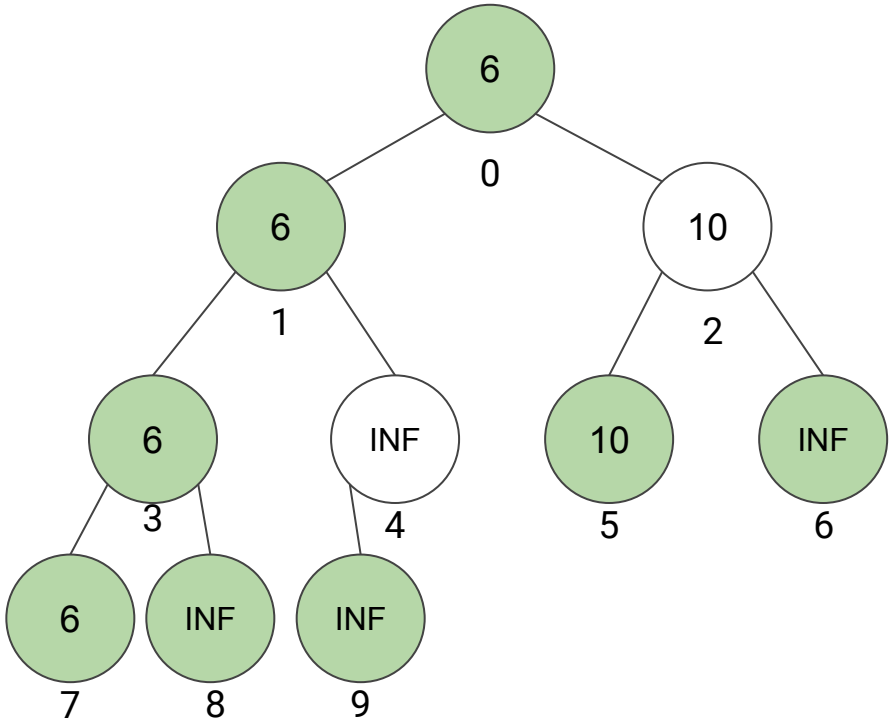
(6)	(7)	(6)	(7)	(9)	10	INF	6	INF	INF
0	1	2	3	4	5	6	7	8	9

SIMULATION



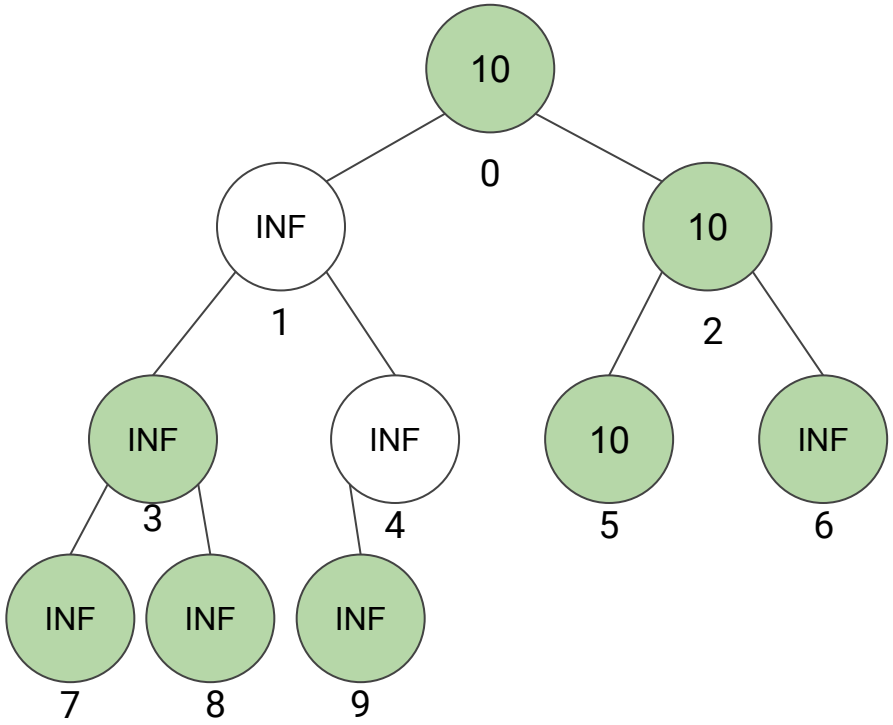
(6)	(7)	(5)	(7)	(9)	10	INF	6	INF	INF
0	1	2	3	4	5	6	7	8	9

SIMULATION



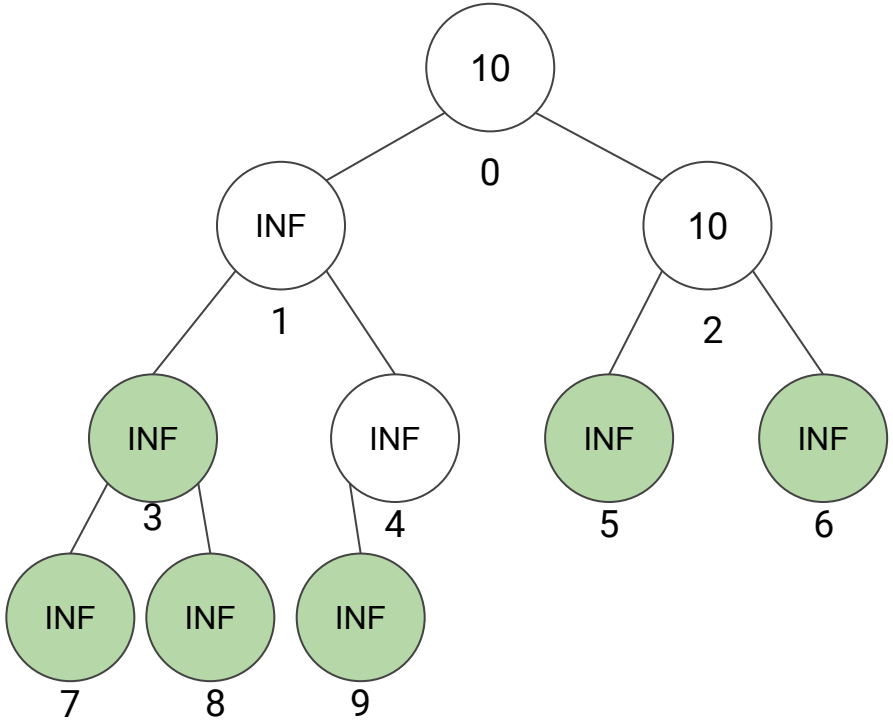
(7)	(7)	(5)	(7)	(9)	10	INF	6	INF	INF
0	1	2	3	4	5	6	7	8	9

SIMULATION



(5)	(7)	(5)	(7)	(9)	10	INF	INF	INF	INF
0	1	2	3	4	5	6	7	8	9

SIMULATION



(5)	(7)	(5)	(7)	(9)	INF	INF	INF	INF	INF
0	1	2	3	4	5	6	7	8	9

SIMULATION

1	2	5	6	10

(5)	(7)	(5)	(7)	(9)	INF	INF	INF	INF	INF
0	1	2	3	4	5	6	7	8	9

INTERNET

```
void tournament_sort() {
  int value;
  create_tree(value);
  for (int i = 0; i < n; i++) {
    a[i] = value;
    recreate(value);
  }
}
```

GROUP

```
void tournament_sort(int data[]){
  int ctr,x;
  Heap tournament_tree = create_tree(data);
  for(ctr=0; ctr<SIZE; ctr++){
          for(x=(tournament_tree.lastNdx)/2-1; x>-1 ; x--){
               play(&tournament_tree, x);
          }
  }

}
```

INTERNET

```
void create_tree(int &value) {
  for (int i = 0; i < n; i++) tmp[n + i] = a[i];
  for (int i = 2 * n - 1; i > 1; i -= 2) {
    int k = i / 2;
    int j = i - 1;
    tmp[k] = winner(i, j);
  }
  value = tmp[tmp[1]];
  tmp[tmp[1]] = INF;
}
```

GROUP

```
Heap create_tree(int data[]) {
     Heap tournament_tree;
     tournament_tree.lastNdx = size*2-2;
     int x,y;

     for(x=0 ; x<SIZE-1 ; x++){
          tournament_tree.elem[x] = INF;
     }


     for(x, y=0 ; y<SIZE*2-1; x++, y++){
          tournament_tree-.elem[x] = data[y];
     }

     return tournament_tree;
}
```

INTERNET

```
int winner(int pos1, int pos2) {
  int u = pos1 >= n ? pos1 : tmp[pos1];
  int v = pos2 >= n ? pos2 : tmp[pos2];
  int retval = tmp[u] <= tmp[v] ? u : v;
  return retval;
}
```

GROUP

```
void play(Heap *L, int parent){
     int left_child = 2*parent + 1;
     int right_child = 2*parent + 2;
     int smallest = left_child;

     if(right_child <= L->lastNdx &&
L->elem[right_child] < L->elem[smallest]){
          smallest = right_child;
     }

     L->elem[parent] = L->elem[smallest];
}
```

INTERNET

```
void recreate(int value) {
  int i = tmp[1];
  while (i > 1) {
    int j;
    int k = i / 2;
    j = (i % 2 == 0 && i < 2 * n - 1) ?
        i + 1 : i - 1;
    tmp[k] = winner(i, j);
    i = k;
  }
  value = tmp[tmp[1]];
  tmp[tmp[1]] = INFINITY;
}
```

GROUP

```
void heapify_subtree(Heap *L, int parent){
     int smallest = parent;
     int left_child = 2*parent + 1;
     int right_child = 2*parent + 2;

     if(left_child <= L->lastNdx && L->elem[left_child] <
L->elem[smallest]){
          smallest = left_child;
     }
     if(right_child <= L->lastNdx && L->elem[right_child]
< L->elem[smallest]){
          smallest = right_child;
     }

     if(smallest != parent){
          L->elem[smallest] = INF;
          heapify_subtree(L, smallest);
     }
}
```

STREAMLINED CODE (INTERNET)

```
#define INF 9999999
int n, a[maxn], tmp[maxn << 1];

int winner(int pos1, int pos2) {
  int u = pos1 >= n ? pos1 : tmp[pos1];
  int v = pos2 >= n ? pos2 : tmp[pos2];
  if (tmp[u] <= tmp[v]) return u;
  return v;
}

 void create_tree(int &value) {
   for (int i = 0; i < n; i++) tmp[n + i] = a[i];
   for (int i = 2 * n - 1; i > 1; i -= 2) {
     int k = i / 2;
     int j = i - 1;
     tmp[k] = winner(i, j);
   }
   value = tmp[tmp[1]];
   tmp[tmp[1]] = INF;
 }
```

```
void recreate(int &value) {
  int i = tmp[1];
  while (i > 1) {
    int j, k = i / 2;
    if (i % 2 == 0 && i < 2 * n - 1)
      j = i + 1;
    else
      j = i - 1;
    tmp[k] = winner(i, j);
    i = k;
  }
  value = tmp[tmp[1]];
  tmp[tmp[1]] = INF;
}

void tournament_sort() {
  int value;
  create_tree(value);
  for (int i = 0; i < n; i++) {
    a[i] = value;
    recreate(value);
  }
}
```

STREAMLINED CODE (GROUP)

```
//definitions and declarations

#define INF 9999999
#define SIZE 5

typedef struct{
     int elem[SIZE*2-1];
     int lastNdx;
}Heap;

int data[SIZE*2-1];
Heap tournament_tree;
```

```
void tournament_sort(Heap *tournament_tree, int data[]){
  int ctr,x;
  create_tree(tournament_tree,data);
  for(ctr=0; ctr<SIZE; ctr++){
          for(x=(tournament_tree.lastNdx)/2-1; x>-1 ; x--){
               play(&tournament_tree, x);
          }
  }

}
```

STREAMLINED CODE (GROUP)

```
void play(Heap *L, int parent){
     int left_child = 2*parent + 1;
     int right_child = 2*parent + 2;
     int smallest = left_child;

     if(right_child <= L->lastNdx &&
L->elem[right_child] < L->elem[smallest]){
          smallest = right_child;
     }

     L->elem[parent] = L->elem[smallest];
}
```

```
void create_tree(Heap *tournament_tree) {
  int x,y, ctr;


     for(x=0 ; x<SIZE-1 ; x++){
          tournament_tree.elem[x] = INF;
     }


     for(x, y=0 ; y<SIZE*2-1; x++, y++){
          tournament_tree.elem[x] = data[y];
     }
}
```

References:

https://en.oi-wiki.org/basic/tournament-sort/
https://www.routledge.com/rsc/downloads/Chapter_3_9781138196186.pdf