2020

Contents

1

- , :

  1.
     ..: 64, 1000
  2.
     ..: $\log_8 n$, $\log_2 n$
  3.
     ..: $4n$, $100n$
  4.
     ..: $n \log_8 n$, $n \log_2 n$
  5.
     ..: $8n^2$, $6n^3$
  6.
     ..: $8^2 n$, $6^3 n$

- n^k c^n; $k >= 1$ $c > 1$ . .

  1. n^k is $\mathcal{O}(c^n)$
  2. n^k is $\Omega(c^n)$
  3. n^k is $\Theta(c^n)$

- $log_2 n$ $log_8 n$; .

  1. $log_2 n$ is $\mathcal{O}(log_8 n)$
  2. $log_2 n$ is $\Omega(log_8 n)$
  3. $log_2 n$ is $\Theta(log_8 n)$

2

2.1

2.1.1 (Binary) -

```
1     int iterativeBinarySearch(int pin[], int low, int high, int num)
2     {
3         int mid;
4         while (low <= high)
5         {
6             mid = (low + high) / 2;
7             if (pin[mid] == num)
8             {
9                 return mid;
10            }
11            if (pin[mid] < num)
12            {
13                low = mid + 1;
14            }
15            else
16            {
17                high = mid - 1;
18            }
19        }
20        return -1;
21    }
```

2.1.2 (Binary) -

```
1     int recursiveBinarySearch(int pin[], int low, int high, int num)
2     {
3         int mid;
4         if (low <= high)
5         {
6             mid = (low + high) / 2;
7             if (pin[mid] == num)
8             {
9                 return mid;
10            }
11            if (pin[mid] < num)
12            {
13                return recursiveBinarySearch(pin, mid+1, high, num)
14            }
15            else
16            {
17                return recursiveBinarySearch(pin, low, mid-1, num)
18            }
19        }
20        return -1;
21    }
```

2.2

2.2.1 (Selection) -

```
1     void selectionSort(int pin[], int size)
2     {
3         int i, j, minPos, temp;
4         for (i = 0; i <= size; j++)
5         {
6             minPos = i;
7             for (j = i + 1; j <= size; j++)
8             {
9                 if (pin[minPos] > pin[j])
10                {
11                    minPos = j;
12                }
13            }
14            if (minPos != i)
15            {
16                temp = pin[i];
17                pin[i] = pin[minPos];
18                pin[minPos] = temp;
19            }
20        }
21    }
```

2.2.2 (Insertion) -

```
1     void insertionSort(int pin[], int size)
2     {
3         int i, j, value;
4         for (i = 1; i <= size; i++)
5         {
6             value = pin[i];
7             j = i - 1;
8             while (j >= 0 && pin[j] > value)
9             {
10                pin[j+1] = pin[j];
11                j--;
12            }
13            pin[j+1] = value;
14        }
15    }
```

2.2.3 (Bubble) -

```
1     void bubbleSort(int pin[], int size)
2     {
3         int i, j, temp;
4         for (i = 1; i <= size; i++)
5         {
6             for (j = size; j >= i; j--)
7             {
8                 if (pin[j] < pin[j-1])
9                 {
10                    temp = pin[j];
11                    pin[j] = pin[j-1];
12                    pin[j-1] = temp;
13                }
14            }
15        }
16    }
```

2.2.4 (Merge) -

```
1     void mergeSort(int pin[], int low, int high)
2     {
3         int mid;
4         if (low < high)
5         {
6             mid = (low + high) / 2;
7             mergeSort(pin, low, mid);
8             mergeSort(pin, mid+1, high);
9             merge(pin, low, mid, mid+1, high);
10        }
11    }
```

2.2.5 (QuickSort) -

```
1     void quickSort(int pin[], int low, int high)
2     {
3         int pivot;
4         if (low < high)
5         {
6             pivot = partition(pin, low, high);
7             quickSort(pin, low, pivot-1);
8             quickSort(pin, pivot+1, high);
9         }
10    }
11
12    int partition(int pin[], int low, int high)
13    {
14        int pivot, smallerPos, i, tmp;
15
16        pivot = pin[high];
17        smallerPos = low - 1;
18
19        for (i = low; i <= high - 1; i++)
20        {
21            if (pin[i] <= pivot)
22            {
23                smallerPos++;
24                tmp = pin[smallerPos];
25                pin[smallerPos] = pin[i];
26                pin[i] = tmp;
27            }
28        }
29
30        tmp = pin[smallerPos + 1];
31        pin[smallerPos + 1] = pin[high];
32        pin[high] = tmp;
33
34        return (smallerPos + 1);
35    }
```

3

3.1

3.1.1 BFS

3.1.2 DFS

3.2

3.3 (MST)

3.3.1 Prim

3.3.2 Kruskal