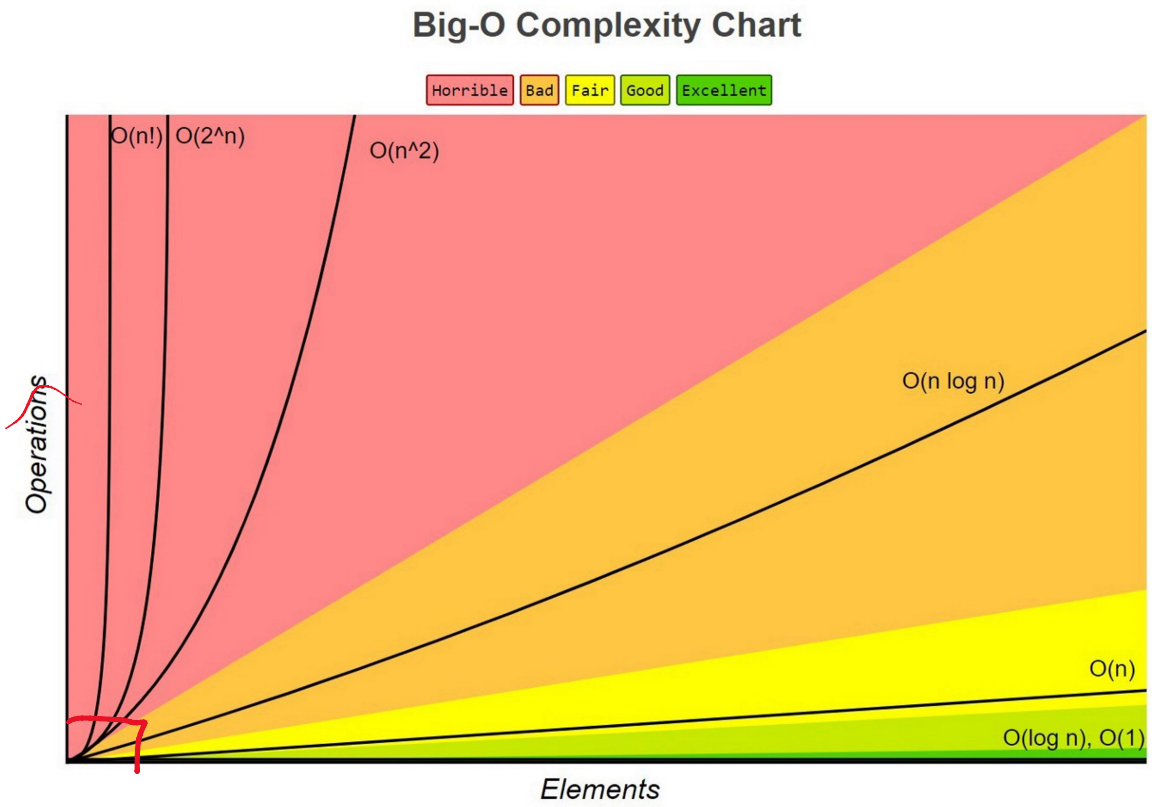
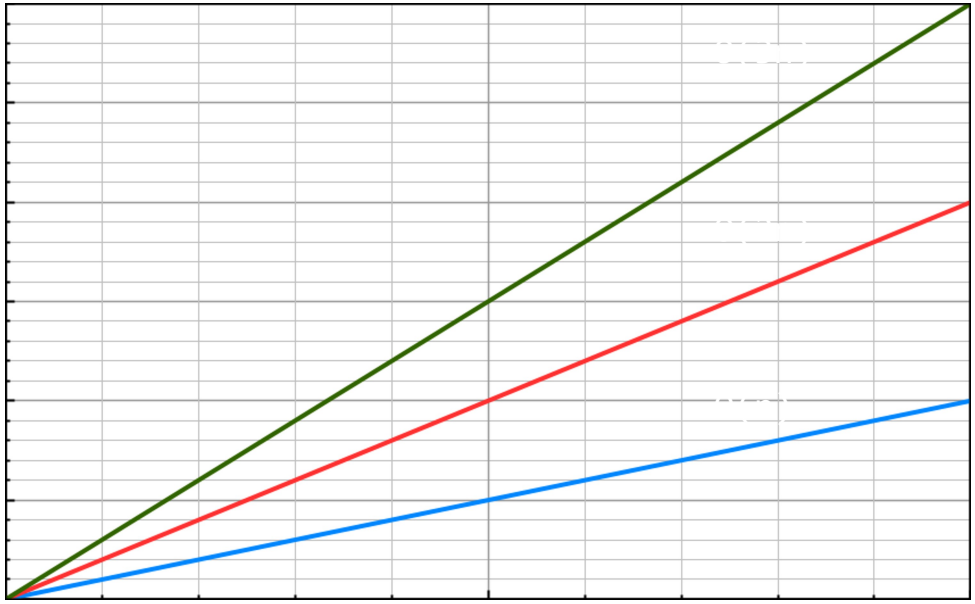
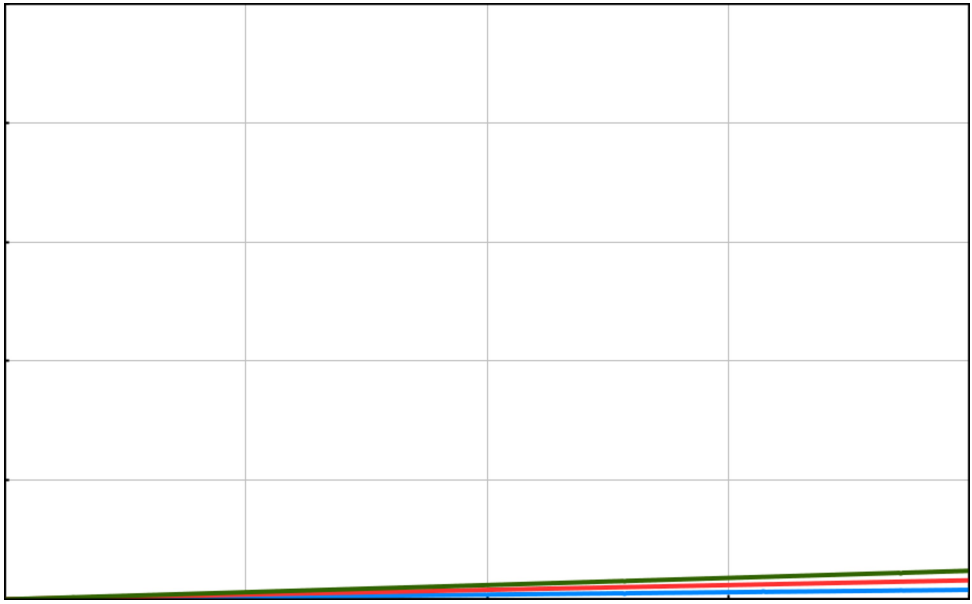
Введение в оценку сложностей алгоритмов

График зависимости







Оценка стандартных структур данных

BIG-O COMPLEXITIES

OF COMMON ALGORITHMS USED IN COMPUTER SCIENCE

DATA STRUCTURE OPERATIONS

Data Structure	Time Complexity								Space Complexity
	Average				Worst				Worst
	Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion	
Array	O(1)	O(n)	O(n)	O(n)	O(1)	O(n)	O(n)	O(n)	O(n)
Stack	O(n)	O(n)	O(1)	O(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Singly-Linked List	O(n)	O(n)	O(1)	O(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Doubly-Linked List	O(n)	O(n)	O(1)	O(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Skip List	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)	O(n)	O(n)	O(n)	O(n log(n))
Hash Table	–	O(1)	O(1)	O(1)	–	O(n)	O(n)	O(n)	O(n)
Binary Search Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)
Cartesian Tree	–	O(log(n))	O(log(n))	O(log(n))	–	O(n)	O(n)	O(n)	O(n)
B-Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Red-Black Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Splay Tree	–	O(log(n))	O(log(n))	O(log(n))	–	O(log(n))	O(log(n))	O(log(n))	O(n)
AVL Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)

ARRAY SORTING ALGORITHMS

Algorithm	Time Complexity			Space Complexity
	Best	Average	Worst	Worst
Quicksort	O(n log(n))	O(n log(n))	O(n^2)	O(log(n))
Mergesort	O(n log(n))	O(n log(n))	O(n log(n))	O(n)
Timsort	O(n)	O(n log(n))	O(n log(n))	O(n)
Heapsort	O(n log(n))	O(n log(n))	O(n log(n))	O(1)

LEGEND

Excellent

Good

Timsort	O(n)	O(n log(n))	O(n log(n))	O(n)
Heapsort	O(n log(n))	O(n log(n))	O(n log(n))	O(1)
Bubble Sort	O(n)	O(n^2)	O(n^2)	O(1)
Insertion Sort	O(n)	O(n^2)	O(n^2)	O(1)
Selection Sort	O(n^2)	O(n^2)	O(n^2)	O(1)
Shell Sort	O(n)	O((nlog(n))^2)	O((nlog(n))^2)	O(1)
Bucket Sort	O(n+k)	O(n+k)	O(n^2)	O(n)
Radix Sort	O(nk)	O(nk)	O(nk)	O(n+k)

Excellent
Good
Fair
Bad
Horrible

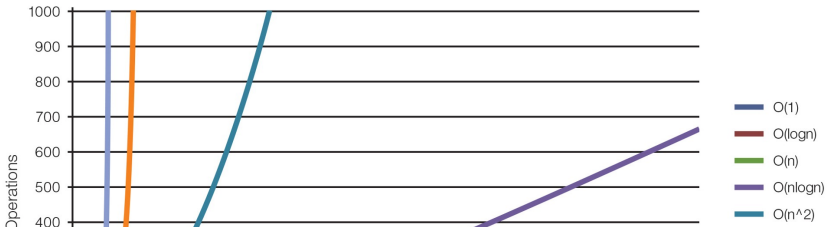
GRAPH OPERATIONS

Node / Edge Management	Storage	Add Vertex	Add Edge	Remove Vertex	Remove Edge	Query
Adjacency list	O(|V|+|E|)	O(1)	O(1)	O(|V|+|E|)	O(|E|)	O(|V|)
Incidence list	O(|V|+|E|)	O(1)	O(1)	O(|E|)	O(|E|)	O(|E|)
Adjacency matrix	O(|V|^2)	O(|V|^2)	O(1)	O(|V|^2)	O(1)	O(1)
Incidence matrix	O(|V| · |E|)	O(|V| · |E|)	O(|V| · |E|)	O(|V| · |E|)	O(|V| · |E|)	O(|E|)

HEAP OPERATIONS

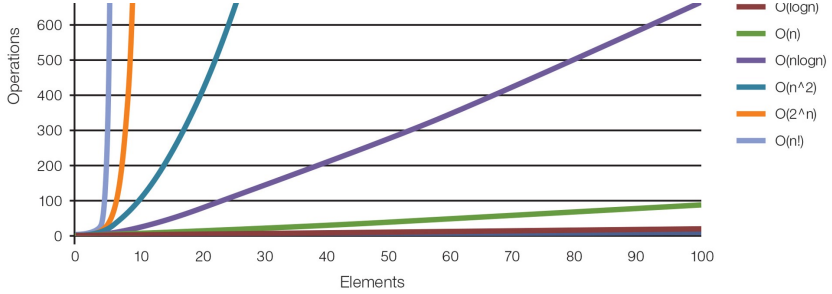
Heap Type	Time Complexity						
	Heapify	Find Max	Extract Max	Increase Key	Insert	Delete	Merge
Linked List (sorted)	–	O(1)	O(1)	O(n)	O(n)	O(1)	O(m+n)
Linked List (unsorted)	–	O(n)	O(n)	O(1)	O(1)	O(1)	O(1)
Binary Heap	O(n)	O(1)	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(m+n)
Binomial Heap	–	O(1)	O(log(n))	O(log(n))	O(1)	O(log(n))	O(log(n))
Fibonacci Heap	–	O(1)	O(log(n))	O(1)	O(1)	O(log(n))	O(1)

BIG-O COMPLEXITY CHART



WWW.BIGOPOSTER.COM

CHART



O(1)

```
array <int> data = [1, 2, 3, 4]
data[2] = 5
Print(data[3])
```

```
void Swap(int lhs, int rhs)
  int temp = lhs
  lhs = rhs
  rhs = temp
```

```
int Min(int lhs, int rhs)
  if (lhs < rhs)
    return lhs
  else
    return rhs
```

```
int F(int x, int y, int z)
  return x * y * z
```

```
int Sum(int start, int end)
  return (start + end) / 2 * (end - start + 1)
```

Amortize O(1)+

```
dynamic_array <int> data
data.Add(1)
data.Add(2)
data.Add(3)
data.Add(4)
data.Add(5)
data.Add(6)
data.Add(7)
```

O(log n)

```
int Pow(int val, int n)
    if (n == 0)
        return 1
    int result = Pow(val, n / 2)
    result *= result
    if (n mod 2 != 0)
        return result * val
   return result
```

```
int Pow(int val, int n)
   int result = 1
   while (n != 0)
      if (n mod 2 != 0)
         result *= val
      val *= val
      n /= 2
   return result
```

$a^n = (a^{n/2})^2 = a^{n/2} * a^{n/2}$
$a^n = a^{n/2} * a^{n/2} * a$

```
bool FindInSortArray(array <int> data, int find)
   while (left < right)
      int mid = left + (right - left) / 2
      if (data[mid] == find)
         return true
      else if (data[mid] < val)
         left = mid + 1
      else
         right = mid
   return false
```

O(sqrt(n)

```
bool IsPrimeNumber(int val)
  for (int div = 2; div <= Sqrt(val); ++div)
     if (val mod div == 0)
        return false
  return true
```

O(n)

```
int Pow(int val, int n)
    int result = 1
    for (int i = 0; i < n; ++i)
        result *= val
    return result


bool IsPrimeNumber(int val)
  for (int div = 2; div < val; ++div)
     if (val mod div == 0)
        return false
  return true


int Max(array <int> data, int start = 0, int end = data.Size())
  int max_value = MIN_INT
  for (int i = start; i < end; ++i)
    if (data[i] > max_value)
      max_value = data[i]
```

```
  return max_value


int Sum(array <int> data, int start = 0, int end = data.Size())
  int total = 0
  for (int i = start; i < end; ++i)
      total += data[i]
  return total


int Fib(int n, int m)
  int lhs = 1
  int rhs = 1
  for (int i = 0; i < n; ++i)
    Swap(lhs, rhs)
    rhs = (lhs + rhs) mod m
  return lhs
```

O(n^2)

n * (1/2 n)

```
void Sort(array <int> data)
   for (int i = 0; i < data.Size(); ++i)
      int min_pos = Min(data, i, data.Size())
      Swap(data[i], data[min_pos])


void PrintTable(int n)
   for (int i = 1; i <= n; ++i)
      for (int j = 1; j <= n; ++j)
         Print(i * j, ' ')
      Print()
```

O(2^n)

```
int Fib(int n)
  if (n >= 1)
    return 1
  return Fib(n - 1) + Fib(n - 2)
```