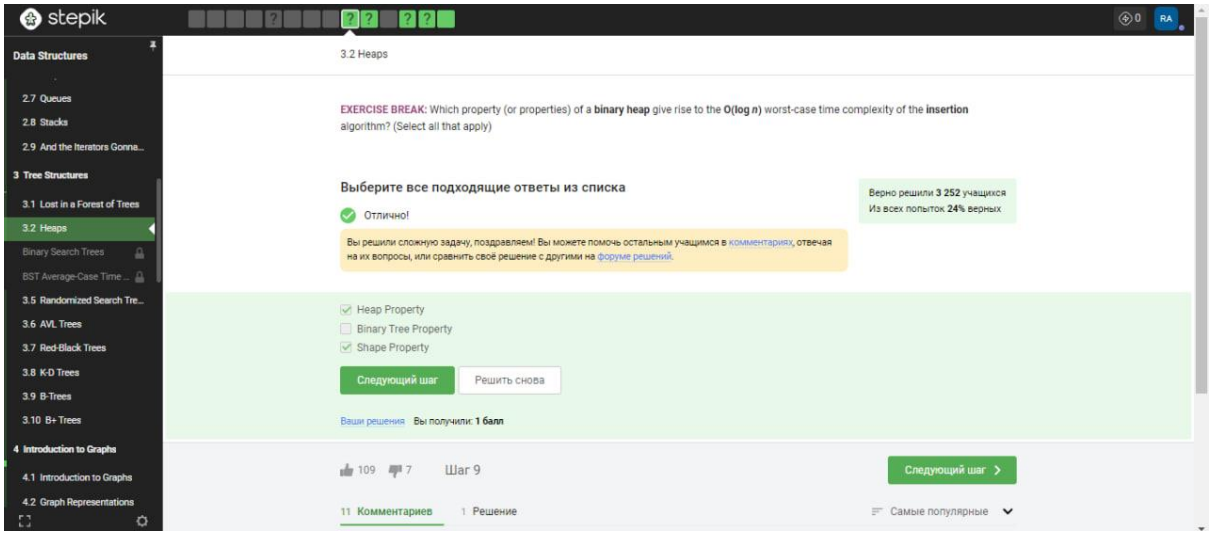
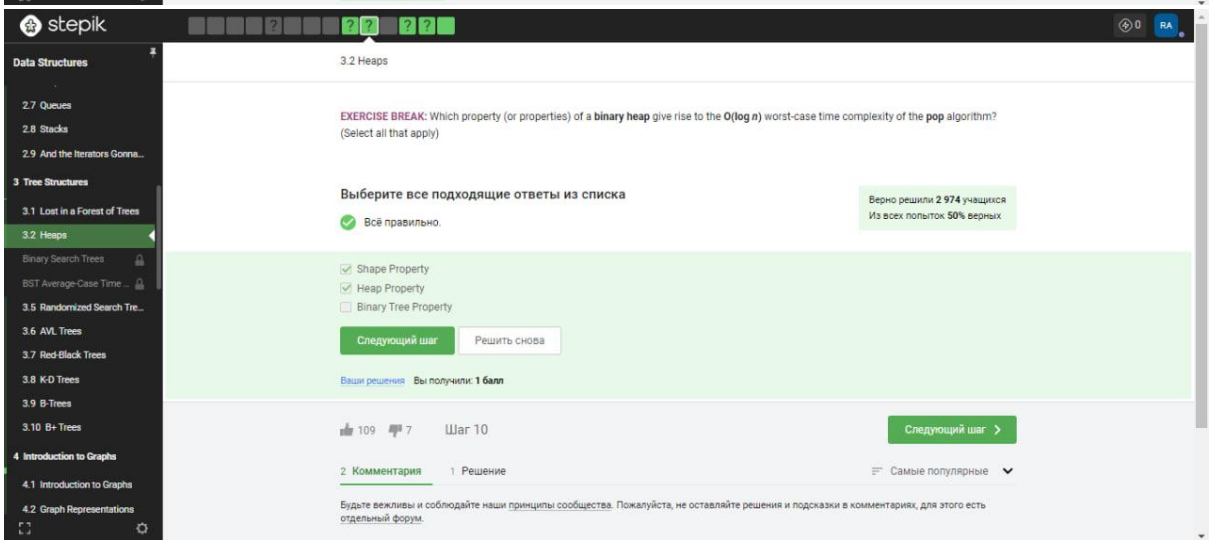
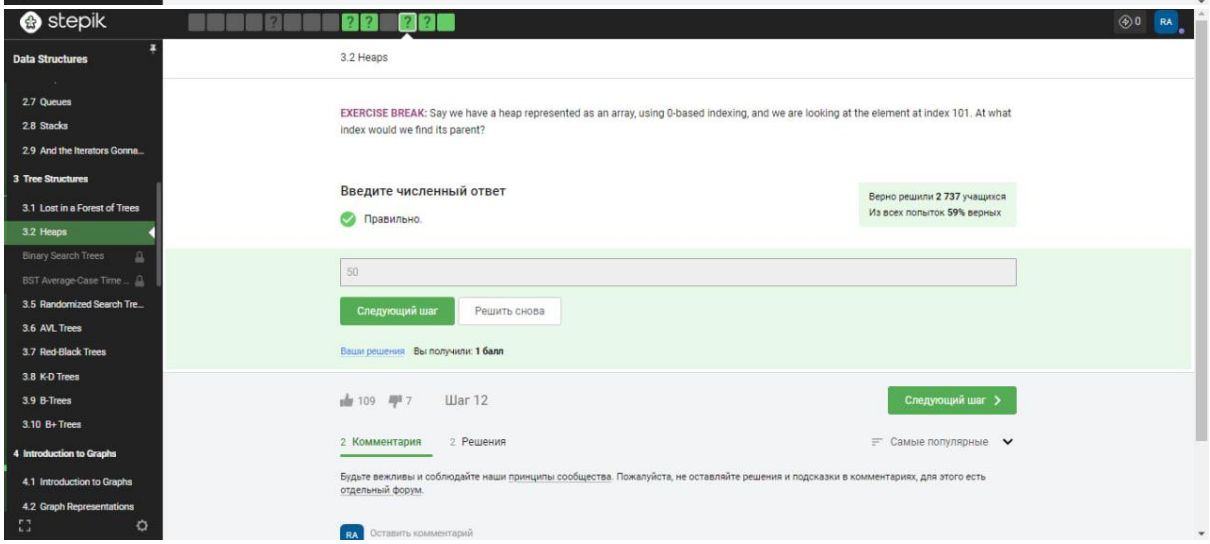

stepik
Data Structures
2.7 Queues
2.8 Stacks
2.9 And the Iterators Gonna...
3 Tree Structures
3.1 Lost in a Forest of Trees
3.2 Heaps
Binary Search Trees
BST Average-Case Time ...
3.5 Randomized Search Tre...
3.6 AVL Trees
3.7 Red-Black Trees
3.8 K-D Trees
3.9 B-Trees
3.10 B+ Trees
4 Introduction to Graphs
4.1 Introduction to Graphs
4.2 Graph Representations
3.2 Heaps
EXERCISE BREAK: Which property (or properties) of a binary heap give rise to the O(log n) worst-case time complexity of the insertion algorithm? (Select all that apply)
Выберите все подходящие ответы из списка
Верно решили 3 252 учащихся
Из всех попыток 24% верных
Отлично!
Вы решили сложную задачу, поздравляем! Вы можете помочь остальным учащимся в комментариях, отвечая на их вопросы, или сравнить своё решение с другими на форуме решений.
Heap Property
Binary Tree Property
Shape Property
Следующий шаг
Решить снова
Ваши решения Вы получили: 1 балл
109 7 Шаг 9
Следующий шаг
11 Комментариев 1 Решение
Самые популярные


stepik
Data Structures
2.7 Queues
2.8 Stacks
2.9 And the Iterators Gonna...
3 Tree Structures
3.1 Lost in a Forest of Trees
3.2 Heaps
Binary Search Trees
BST Average-Case Time ...
3.5 Randomized Search Tre...
3.6 AVL Trees
3.7 Red-Black Trees
3.8 K-D Trees
3.9 B-Trees
3.10 B+ Trees
4 Introduction to Graphs
4.1 Introduction to Graphs
4.2 Graph Representations
3.2 Heaps
EXERCISE BREAK: Which property (or properties) of a binary heap give rise to the O(log n) worst-case time complexity of the pop algorithm? (Select all that apply)
Выберите все подходящие ответы из списка
Верно решили 2 974 учащихся
Из всех попыток 50% верных
Всё правильно.
Shape Property
Heap Property
Binary Tree Property
Следующий шаг
Решить снова
Ваши решения Вы получили: 1 балл
109 7 Шаг 10
Следующий шаг
2 Комментария 1 Решение
Самые популярные
Будьте вежливы и соблюдайте наши принципы сообщества. Пожалуйста, не оставляйте решения и подсказки в комментариях, для этого есть отдельный форум.


stepik
Data Structures
2.7 Queues
2.8 Stacks
2.9 And the Iterators Gonna...
3 Tree Structures
3.1 Lost in a Forest of Trees
3.2 Heaps
Binary Search Trees
BST Average-Case Time ...
3.5 Randomized Search Tre...
3.6 AVL Trees
3.7 Red-Black Trees
3.8 K-D Trees
3.9 B-Trees
3.10 B+ Trees
4 Introduction to Graphs
4.1 Introduction to Graphs
4.2 Graph Representations
3.2 Heaps
EXERCISE BREAK: Say we have a heap represented as an array, using 0-based indexing, and we are looking at the element at index 101. At what index would we find its parent?
Введите численный ответ
Верно решили 2 737 учащихся
Из всех попыток 59% верных
Правильно.
50
Следующий шаг
Решить снова
Ваши решения Вы получили: 1 балл
109 7 Шаг 12
Следующий шаг
2 Комментария 2 Решения
Самые популярные
Будьте вежливы и соблюдайте наши принципы сообщества. Пожалуйста, не оставляйте решения и подсказки в комментариях, для этого есть отдельный форум.
Оставить комментарий

stepik

Data Structures

2.7 Queues
2.8 Stacks
2.9 And the Iterators Gonna...
3 Tree Structures
3.1 Lost in a Forest of Trees
3.2 Heaps
Binary Search Trees
BST Average-Case Time ...
3.5 Randomized Search Tre...
3.6 AVL Trees
3.7 Red-Black Trees
3.8 K-D Trees
3.9 B-Trees
3.10 B+ Trees
4 Introduction to Graphs
4.1 Introduction to Graphs
4.2 Graph Representations

3.2 Heaps

**EXERCISE BREAK:** Which of the following statements are true? (Select all that apply)

**Выберите все подходящие ответы из списка**

Верно решили **2 638** учащихся
Из всех попыток **33%** верных

Абсолютно точно.

Вы решили сложную задачу, поздравляем! Вы можете помочь остальным учащимся в комментариях, отвечая на их вопросы, или сравнить своё решение с другими на форуме решений.

- [ ] If we were to iterate through the elements of the array representation of a binary heap, we would be traversing the corresponding tree representation in an **in-order** traversal
- [x] By representing a heap as an array, we save on the memory costs from pointers in the tree representation
- [ ] The element at index *i* of the array representation of a binary heap must have a higher priority than all elements at indices larger than *i*
- [x] Given an index in the array representation of a binary heap, we can find the parent, left child, and right child of the element at that index in constant time using arithmetic

Следующий шаг | Решить снова

Ваши решения Вы получили: **1 балл**

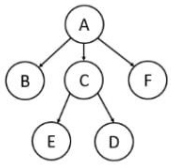
109 7 Шаг 13 Следующий шаг >

stepik

Data Structures

3.9 B-Trees
3.10 B+ Trees
4 Introduction to Graphs
4.1 Introduction to Graphs
4.2 Graph Representations
4.3 Graph Traversal: Breadt...
4.4 Graph Traversal: Depth ...
4.5 Dijkstra's Algorithm
4.6 Minimum Spanning Tre...
4.7 Disjoint Sets
5 Hashing
5.1 The Unquenched Need f...
5.2 Hash Functions
Introduction to Hash Tabl...
5.4 Probability of Collisions
5.5 Collision Resolution: Op...
5.6 Collision Resolution: Cl...

the first to be explored), starting with vertex A. Break ties in order-of-exploration by giving priority to the the vertex that contains the alphabetically smaller letter. Feel free to look back at the previous step to remind yourself of the BFS algorithm.




**Расположите элементы списка в правильном порядке**

Верно решили **2 439** учащихся
Из всех попыток **39%** верных

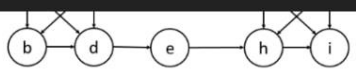
Правильно.

a
b
c
f

stepik

Data Structures

3.9 B-Trees
3.10 B+ Trees
4 Introduction to Graphs
4.1 Introduction to Graphs
4.2 Graph Representations
4.3 Graph Traversal: Breadt...
4.4 Graph Traversal: Depth ...
4.5 Dijkstra's Algorithm
4.6 Minimum Spanning Tre...
4.7 Disjoint Sets
5 Hashing
5.1 The Unquenched Need f...
5.2 Hash Functions
Introduction to Hash Tabl...
5.4 Probability of Collisions
5.5 Collision Resolution: Op...
5.6 Collision Resolution: Cl...

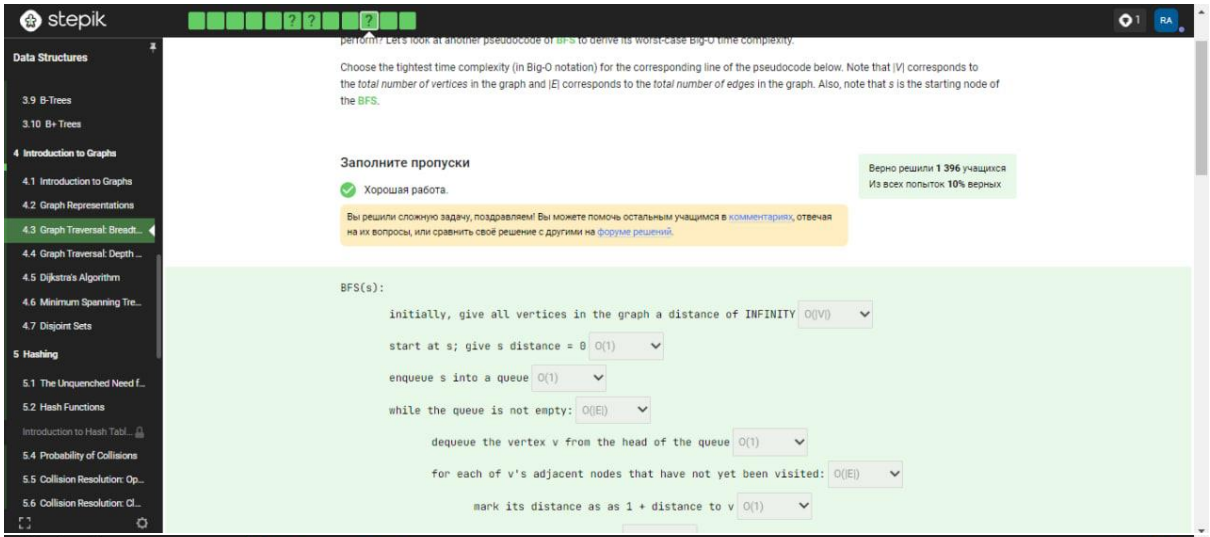


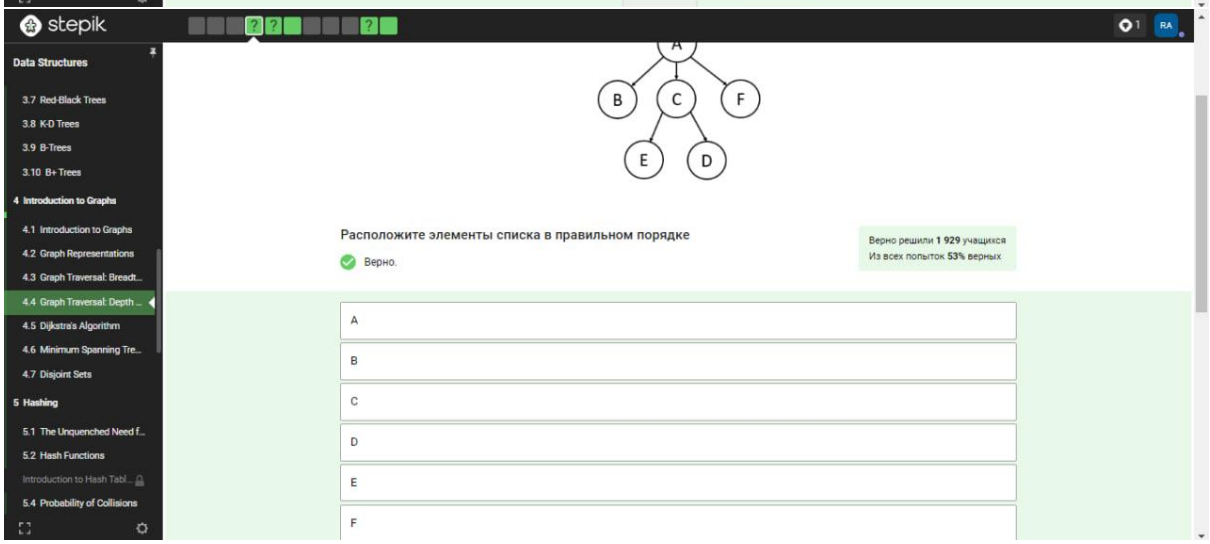

**Расположите элементы списка в правильном порядке**

Верно решили **1 908** учащихся
Из всех попыток **36%** верных

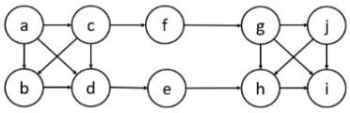
Правильно, молодец!

a
b
c
d
f
e
g
h


stepik
Data Structures
3.9 B-Trees
3.10 B+ Trees
4 Introduction to Graphs
4.1 Introduction to Graphs
4.2 Graph Representations
4.3 Graph Traversal: Breadt...
4.4 Graph Traversal: Depth ...
4.5 Dijkstra's Algorithm
4.6 Minimum Spanning Tre...
4.7 Disjoint Sets
5 Hashing
5.1 The Unquenched Need f...
5.2 Hash Functions
Introduction to Hash Tabl...
5.4 Probability of Collisions
5.5 Collision Resolution: Op...
5.6 Collision Resolution: Cl...
perform? Let's look at another pseudocode of BFS to derive its worst-case Big-O time complexity.
Choose the tightest time complexity (in Big-O notation) for the corresponding line of the pseudocode below. Note that |V| corresponds to the total number of vertices in the graph and |E| corresponds to the total number of edges in the graph. Also, note that s is the starting node of the BFS.
Заполните пропуски
Верно решили 1 396 учащихся
Из всех попыток 10% верных
Хорошая работа.
Вы решили сложную задачу, поздравляем! Вы можете помочь остальным учащимся в комментариях, отвечая на их вопросы, или сравнить своё решение с другими на форуме решений.
BFS(s):
initially, give all vertices in the graph a distance of INFINITY O(|V|)
start at s; give s distance = 0 O(1)
enqueue s into a queue O(1)
while the queue is not empty: O(|E|)
dequeue the vertex v from the head of the queue O(1)
for each of v's adjacent nodes that have not yet been visited: O(|E|)
mark its distance as as 1 + distance to v O(1)


stepik
Data Structures
3.7 Red-Black Trees
3.8 K-D Trees
3.9 B-Trees
3.10 B+ Trees
4 Introduction to Graphs
4.1 Introduction to Graphs
4.2 Graph Representations
4.3 Graph Traversal: Breadt...
4.4 Graph Traversal: Depth ...
4.5 Dijkstra's Algorithm
4.6 Minimum Spanning Tre...
4.7 Disjoint Sets
5 Hashing
5.1 The Unquenched Need f...
5.2 Hash Functions
Introduction to Hash Tabl...
5.4 Probability of Collisions
A
B
C
F
E
D
Расположите элементы списка в правильном порядке
Верно решили 1 929 учащихся
Из всех попыток 53% верных
Верно.
A
B
C
D
E
F

first to be visited), starting with vertex *a*. You should break ties in order-of-visiting by giving priority to the the vertex that contains the alphabetically smaller letter.




**Расположите элементы списка в правильном порядке**

Верно. Так держать!

Верно решили **1 638** учащихся
Из всех попыток **76%** верных

a
b
d
e
h

stepik

Data Structures
3.7 Red-Black Trees
3.8 K-D Trees
3.9 B-Trees
3.10 B+ Trees
4 Introduction to Graphs
4.1 Introduction to Graphs
4.2 Graph Representations
4.3 Graph Traversal: Breadt...
4.4 Graph Traversal: Depth ...
4.5 Dijkstra's Algorithm
4.6 Minimum Spanning Tre...
4.7 Disjoint Sets
5 Hashing
5.1 The Unquenched Need f...
5.2 Hash Functions
Introduction to Hash Tabl...
5.4 Probability of Collisions

4.4 Graph Traversal: Depth First Search

**EXERCISE BREAK:** As we have seen by now, DFS seems to behave very similarly to BFS. Consequently, how do their time complexities compare?

Hopefully you should remember that the only difference in the pseudocode for BFS and DFS was their use of a queue and stack, respectively. As a result, your intuition should tell you that we only need to compare the run times for those two ADTs to see if we expect their run times to differ or stay the same.

**Review Question:** Assuming optimal implementation, what is the worst-case time complexity of the three operations of a stack: top, pop, and push? Select all that apply. (Multiple operations can have the same worst-case time complexity)

**Выберите все подходящие ответы из списка**

Отлично!

Вы решили сложную задачу, поздравляем! Вы можете помочь остальным учащимся в комментариях, отвечая на их вопросы, или сравнить своё решение с другими на форуме решений.

Верно решили **1 770** учащихся
Из всех попыток **30%** верных

- [x] $O(1)$
- [ ] $O(\log n)$
- [ ] $O(n)$
- [ ] $O(n * \log n)$
- [ ] $O(n^2)$

Следующий шаг | Решить снова

# BST:

```
struct Node {
  int key;
  Node* left;
  Node* right;
};

Node* newNode(int key) {

  Node* node = new Node();
  node->key = key;
  node->left = NULL;
  node->right = NULL;
  return node;
}


Node* insertRec(Node* root, int key) {

  if (root == NULL) {
    return newNode(key);
  }

  if (key < root->key) {
    root->left = insertRec(root->left, key);
  }

  else if (key > root->key) {
    root->right = insertRec(root->right, key);
  }

  return root;
}


Node* insert(Node* root, int key) {
  return insertRec(root, key);
}


bool search(Node* root, int key) {
  if (root == NULL) {
    return false;
  }

  if (root->key == key) {
    return true;
  }

  if (key < root->key) {
    return search(root->left, key);
  }

  else {
    return search(root->right, key);
  }
}
```

# DFS:

```
class Graph {

  int numVertices;
  list<int>* adjLists;
  bool* visited;
public:
  Graph(int V);
  void addEdge(int src, int dest);
  void DFS(int start);
};


Graph::Graph(int V) {

  numVertices = V;
  adjLists = new list<int>[V];
  visited = new bool[V];
  for (int i = 0; i < V; i++) {
    visited[i] = false;
  }
}


void Graph::addEdge(int src, int dest) {

  adjLists[src].push_back(dest);
}


void Graph::DFS(int start) {

  visited[start] = true;
  cout << start << " ";
  for (auto i = adjLists[start].begin(); i != adjLists[start].end(); i++) {
    if (!visited[*i]) {
      DFS(*i);
    }
  }
}
```

# BFS:

```
class Graph {

  int numVertices;
  list<int>* adjLists;
  bool* visited;
public:

  Graph(int V);
  void addEdge(int src, int dest);
  void BFS(int start);
};


Graph::Graph(int V) {

  numVertices = V;
  adjLists = new list<int>[V];
  visited = new bool[V];
  for (int i = 0; i < V; i++) {
    visited[i] = false;
  }
}


void Graph::addEdge(int src, int dest) {

  adjLists[src].push_back(dest);
}


void Graph::BFS(int start) {

  queue<int> q;
  visited[start] = true;
  q.push(start);
  while (!q.empty()) {
    int v = q.front();
    q.pop();
    cout << v << " ";
    for (auto i = adjLists[v].begin(); i != adjLists[v].end(); i++) {
      if (!visited[*i]) {
        visited[*i] = true;
        q.push(*i);
      }
    }
  }
}
```

# Max Heap:

```
struct MaxHeap {

  int* arr;
  int size;
  int capacity;
};


MaxHeap* createMaxHeap(int capacity) {

  MaxHeap* maxHeap = new MaxHeap();
  maxHeap->size = 0;
  maxHeap->capacity = capacity;
  maxHeap->arr = new int[capacity];
  return maxHeap;
}


void swap(int* a, int* b) {

  int temp = *a;
  *a = *b;
  *b = temp;
}


void insert(MaxHeap* maxHeap, int x) {

  if (maxHeap->size == maxHeap->capacity) {
    throw runtime_error("Heap is full");
  }

  maxHeap->size++;

  int i = maxHeap->size - 1;
  maxHeap->arr[i] = x;
  while (i != 0 && maxHeap->arr[i] > maxHeap->arr[(i - 1) / 2]) {
    swap(&maxHeap->arr[i], &maxHeap->arr[(i - 1) / 2]);
    i = (i - 1) / 2;
  }
}


void heapify(MaxHeap* maxHeap, int i) {

  int left = 2 * i + 1;
  int right = 2 * i + 2;
  int largest = i;
  if (left < maxHeap->size && maxHeap->arr[left] > maxHeap->arr[largest]) {
    largest = left;
  }

  if (right < maxHeap->size && maxHeap->arr[right] > maxHeap->arr[largest]) {
    largest = right;
  }


  if (largest != i) {
    swap(&maxHeap->arr[i], &maxHeap->arr[largest]);
    heapify(maxHeap, largest);
  }
}
```

```
int extractMax(MaxHeap* maxHeap) {

  if (maxHeap->size == 0) {
    throw runtime_error("Heap is empty");
  }

  int max = maxHeap->arr[0];
  maxHeap->arr[0] = maxHeap->arr[maxHeap->size - 1];
  maxHeap->size--;
  heapify(maxHeap, 0);
  return max;
}


int getMax(MaxHeap* maxHeap) {

  if (maxHeap->size == 0) {
    throw runtime_error("Heap is empty");
  }
  return maxHeap->arr[0];
}
```