**Please evaluate based on your preferences. You can also use the following rubric for marking.**

Heap Quiz: Set - A

**Rubric:**

| **SN** | **Criteria** | **Marks** |
| --- | --- | --- |
| 1 | Finding out which Heap to Use | 1 |
| 2 | Creating a heap from an array (loop/function) | 2 |
| 3 | Create a Result Heap (Same Size as Heap/Other Size) | 1 |
| 4 | Extract Min Function | 3 |
| 5 | Sink Function | 3 |
| 6 | Proper use of Conditions (x, y checking + discarding unwanted values) | 3 |
| 7 | Proper use of Insert + Extract Min in result heap | 2 |
|  | **Total:** | **15** |

**Code:**

| **#! Set-A-Tentative-Solution (Python)**   | **def parentIndex(index):  if index == 0 :  return 0  return (index-1) // 2  def leftIndex(index):  return (index\*2)+1  def rightIndex(index):  return (index\*2)+2  class MinHeap:   def \_\_init\_\_(self, capacity):  self.heap = [0] \* capacity  self.capacity = capacity  self.size = 0     def insert(self, item):  self.heap[self.size] = item  self.swim(self.size)  self.size += 1    def swim(self, index):  item,parent\_index = self.heap[index],parentIndex(index)  if item < self.heap[parent\_index]:  self.heap[parent\_index], self.heap[index] = self.heap[index],self.heap[parent\_index]  self.swim(parent\_index)   def extractMin(self):   if self.size == 0:  return None  item = self.heap[0]  self.heap[0] = self.heap[self.size-1] #! Why -1   self.size -= 1  self.sink(0)  self.heap[self.size] = None  return item    def sink(self, index):  min\_index = index  item, left\_index, right\_index = self.heap[index], leftIndex(index), rightIndex(index)    #! Check Left Child  if left\_index < self.size and self.heap[left\_index] < self.heap[min\_index]:  min\_index = left\_index    #! Check right child  if right\_index < self.size and self.heap[right\_index] < self.heap[min\_index]:  min\_index = right\_index    if self.heap[index] > self.heap[min\_index] and min\_index != index:  self.heap[index], self.heap[min\_index] = self.heap[min\_index], self.heap[index]  self.sink(min\_index)    def create\_heap\_from\_array(self, arr):  for i in range(len(arr)):  self.insert(arr[i])  def print\_heap(self):  for i in range(self.size):  print(self.heap[i], end=" ")   def min\_k\_elements(self, x, y):   size = y - x + 1  #-Same Size or Calculated-#   size = len(self.heap)   result = MinHeap(size)   for i in range(1, len(self.heap)+1):  #? If the index is between x and y, then insert the value  if i >= x and i <= y:   result.insert(self.extractMin())  #? If the index is not between x and y, then extract min and discard  else:  self.extractMin()  return result x = 2  y = 5  array = [11, 15, 8, 2, 31, 23] initial\_heap = MinHeap(6) initial\_heap.create\_heap\_from\_array(array) print("Initial Heap:", initial\_heap.heap) result\_heap = initial\_heap.min\_k\_elements(x,y) print(f"New heap with elements from position {x} to {y}:") result\_heap.print\_heap()** | | --- | |
| --- | --- |

| **#! Set-A-Tentative-Solution (Java)**   | **public class MinHeap {  private int[] heap;  private int capacity;  private int size;    public MinHeap(int capacity) {  this.heap = new int[capacity];  this.capacity = capacity;  this.size = 0;  }    private static int parentIndex(int index) {  if (index == 0) {  return 0;  }  return (index - 1) / 2;  }    private static int leftIndex(int index) {  return (index \* 2) + 1;  }    private static int rightIndex(int index) {  return (index \* 2) + 2;  }    public void insert(int item) {  heap[size] = item;  swim(size);  size++;  }    private void swim(int index) {  int item = heap[index];  int parentIndex = parentIndex(index);  if (item < heap[parentIndex]) {  int temp = heap[parentIndex];  heap[parentIndex] = heap[index];  heap[index] = temp;  swim(parentIndex);  }  }    public Integer extractMin() {  if (size == 0) {  return null;  }  int item = heap[0];  heap[0] = heap[size - 1];  size--;  sink(0);  heap[size] = 0;  return item;  }    private void sink(int index) {  int minIndex = index;  int leftIndex = leftIndex(index);  int rightIndex = rightIndex(index);    if (leftIndex < size && heap[leftIndex] < heap[minIndex]) {  minIndex = leftIndex;  }    if (rightIndex < size && heap[rightIndex] < heap[minIndex]) {  minIndex = rightIndex;  }    if (heap[index] > heap[minIndex] && minIndex != index) {  int temp = heap[index];  heap[index] = heap[minIndex];  heap[minIndex] = temp;  sink(minIndex);  }  }    public void createHeapFromArray(int[] arr) {  for (int i = 0; i < arr.length; i++) {  insert(arr[i]);  }  }    public MinHeap minKElements(int x, int y) {  int size = y - x + 1;  MinHeap result = new MinHeap(size);    // Extract and discard elements before x  for (int i = 1; i < x; i++) {  extractMin();  }    // Extract and keep elements from x to y  for (int i = x; i <= y; i++) {  Integer minVal = extractMin();  if (minVal != null) {  result.insert(minVal);  }  }    return result;  }    public void printHeap() {  System.out.print("[");  for (int i = 0; i < size; i++) {  System.out.print(heap[i]);  if (i < size - 1) {  System.out.print(", ");  }  }  System.out.println("]");  }    public static void main(String[] args) {  int x = 2;  int y = 5;  int[] array = new int[]{11, 15, 8, 2, 31, 23};    MinHeap initialHeap = new MinHeap(6);  initialHeap.createHeapFromArray(array);    System.out.print("Initial Heap: ");  initialHeap.printHeap();    MinHeap resultHeap = initialHeap.minKElements(x, y);  System.out.printf("New heap with elements from position %d to %d:%n", x, y);  resultHeap.printHeap();  } }** | | --- | |
| --- | --- |

**Same as Set-A but MaxHeap instead of MinHeap**

Heap Quiz: Set - B

**Rubric:**

| **SN** | **Criteria** | **Marks** |
| --- | --- | --- |
| 1 | Finding out which Heap to Use | 1 |
| 2 | Creating a heap from an array (loop/function) | 2 |
| 3 | Create a Result Heap (Same Size as Heap/Other Size) | 1 |
| 4 | Extract Max Function | 3 |
| 5 | Sink Function | 3 |
| 6 | Proper use of Conditions (x, y checking + discarding unwanted values) | 3 |
| 7 | Proper use of Insert + Extract Max in result heap | 2 |
|  | **Total:** | **15** |

**Code:**

| **#! Set-B-Tentative-Solution**   | **def parentIndex(index):  if index == 0:  return 0  return (index-1) // 2  def leftIndex(index):  return (index\*2)+1  def rightIndex(index):  return (index\*2)+2  class MaxHeap:  def \_\_init\_\_(self, capacity):  self.heap = [0] \* capacity  self.capacity = capacity  self.size = 0   def insert(self, item):  self.heap[self.size] = item  self.swim(self.size)  self.size += 1    def swim(self, index):  item, parent\_index = self.heap[index], parentIndex(index)  if item > self.heap[parent\_index]:  self.heap[parent\_index], self.heap[index] = self.heap[index], self.heap[parent\_index]  self.swim(parent\_index)   def extractMax(self):  if self.size == 0:  return None  item = self.heap[0]  self.heap[0] = self.heap[self.size-1]  self.size -= 1  self.sink(0)  self.heap[self.size] = None  return item    def sink(self, index):  max\_index = index  item, left\_index, right\_index = self.heap[index], leftIndex(index), rightIndex(index)    if left\_index < self.size and self.heap[left\_index] > self.heap[max\_index]:  max\_index = left\_index    if right\_index < self.size and self.heap[right\_index] > self.heap[max\_index]:  max\_index = right\_index    if self.heap[index] < self.heap[max\_index] and max\_index != index:  self.heap[index], self.heap[max\_index] = self.heap[max\_index], self.heap[index]  self.sink(max\_index)   def create\_heap\_from\_array(self, arr):  for i in range(len(arr)):  self.insert(arr[i])    def get\_elements\_between\_positions(self, x, y):  size = y - x + 1  #! OR   size = len(self.heap)  result = MaxHeap(size)  for i in range(1, len(self.heap)+1):  if i >= x and i <= y:  result.insert(self.extractMax())  else:  self.extractMax()    return result   x = 2  y = 5  array = [11, 15, 8, 2, 31, 23] initial\_heap = MaxHeap(6) initial\_heap.create\_heap\_from\_array(array) print("Initial Heap:", initial\_heap.heap) result\_heap = initial\_heap.get\_elements\_between\_positions(x, y) print(f"New heap with elements from position {x} to {y}:") result\_heap.print\_heap()** | | --- | |
| --- | --- |

| **#! Set-A-Tentative-Solution (Java)**   | **public class MaxHeap {  private int[] heap;  private int capacity;  private int size;    public MaxHeap(int capacity) {  this.heap = new int[capacity];  this.capacity = capacity;  this.size = 0;  }    private static int parentIndex(int index) {  if (index == 0) {  return 0;  }  return (index - 1) / 2;  }    private static int leftIndex(int index) {  return (index \* 2) + 1;  }    private static int rightIndex(int index) {  return (index \* 2) + 2;  }    public void insert(int item) {  heap[size] = item;  swim(size);  size++;  }    private void swim(int index) {  int item = heap[index];  int parentIndex = parentIndex(index);  if (item > heap[parentIndex]) {  int temp = heap[parentIndex];  heap[parentIndex] = heap[index];  heap[index] = temp;  swim(parentIndex);  }  }    public Integer extractMax() {  if (size == 0) {  return null;  }  int item = heap[0];  heap[0] = heap[size - 1];  size--;  sink(0);  heap[size] = 0;  return item;  }    private void sink(int index) {  int maxIndex = index;  int leftIndex = leftIndex(index);  int rightIndex = rightIndex(index);    if (leftIndex < size && heap[leftIndex] > heap[maxIndex]) {  maxIndex = leftIndex;  }    if (rightIndex < size && heap[rightIndex] > heap[maxIndex]) {  maxIndex = rightIndex;  }    if (heap[index] < heap[maxIndex] && maxIndex != index) {  int temp = heap[index];  heap[index] = heap[maxIndex];  heap[maxIndex] = temp;  sink(maxIndex);  }  }    public void createHeapFromArray(int[] arr) {  for (int i = 0; i < arr.length; i++) {  insert(arr[i]);  }  }    public MaxHeap getElementsBetweenPositions(int x, int y) {  int size = y - x + 1;  MaxHeap result = new MaxHeap(size);    // Extract and discard elements before x  for (int i = 1; i < x; i++) {  extractMax();  }    // Extract and keep elements from x to y  for (int i = x; i <= y; i++) {  Integer maxVal = extractMax();  if (maxVal != null) {  result.insert(maxVal);  }  }    return result;  }    public void printHeap() {  System.out.print("[");  for (int i = 0; i < size; i++) {  System.out.print(heap[i]);  if (i < size - 1) {  System.out.print(", ");  }  }  System.out.println("]");  }    public static void main(String[] args) {  int x = 2;  int y = 5;  int[] array = new int[]{11, 15, 8, 2, 31, 23};    MaxHeap initialHeap = new MaxHeap(6);  initialHeap.createHeapFromArray(array);    System.out.print("Initial Heap: ");  initialHeap.printHeap();    MaxHeap resultHeap = initialHeap.getElementsBetweenPositions(x, y);  System.out.printf("New heap with elements from position %d to %d:%n", x, y);  resultHeap.printHeap();  } }** | | --- | |
| --- | --- |