**SET A**

| public class MinHeap {  // Extract Min function  public int extractMin() {  if (heapSize == 0) {  throw new IllegalStateException("Heap underflow");  }    int minValue = arr[1];    arr[1] = arr[heapSize];  arr[heapSize] = 0;  heapSize--;    sink(1);  return minValue;  }  // Sink function  private void sink(int index) {  while (2 \* index <= heapSize) {  int child = 2 \* index;  if (child < heapSize && arr[child] > arr[child + 1]) {  child++;  }  if (arr[index] <= arr[child]) {  break;  }  // Swap with the smaller child  int temp = arr[index];  arr[index] = arr[child];  arr[child] = temp;  index = child;  }  }  // minOperation method  public int minOperation() {  int operations = 0;  while (heapSize > 0) {  int minVal = extractMin();  int originalSize = heapSize;  for (int i = 1; i <= originalSize; i++) {  arr[i] -= minVal;  }  operations++;  }  return operations;  }  } |
| --- |

**SET B**

| // Min operation: floor divide all remaining non-zero values by the extracted minimum  public int minOperation() {  int operations = 0;  while (heapSize > 0) {  int minVal = extractMin();    int originalSize = heapSize;  for (int i = 1; i <= originalSize; i++) {  if (arr[i] > 0) {  arr[i] = arr[i] / minVal;  }  }  operations++;  }  return operations;  } |
| --- |

**Rest Is similar to SET A**

| **SL** | **Points to meet** | **Marks (15)** |
| --- | --- | --- |
| 1 | In the sink function, accurate child index calculation and comparison | 3 |
| 2 | In the sink function, proper swapping mechanism and loop termination | 3 |
| 3 | Heap emptiness check | 1 |
| 4 | Heap reorganization | 2 |
| 5 | Correct loop structure and termination | 2 |
| 6. | Correct Calculation (subtraction for set A and floor division conditions for set B ) | 3 |
| 7. | Correct Output return | 1 |
| **Total** | | **15** |