Algorithm:

```
Algorithm RandomizedQuickSort(A, low, high){
   if (low < high) then{
      pivot_index := RandomPartition(A, low, high);
      RandomizedQuickSort(A, low, pivot_index - 1);
      RandomizedQuickSort(A, pivot_index + 1, high);
      }
}
Algorithm  RandomPartition(A, low, high){
   pivot_index := Random(low, high);
   swap (A[pivot_index], A[high]);
   return Partition(A, low, high);
}
Algorithm  Partition(A, low, high){
   pivot := A[high];
   i := low - 1;
   for j := low to high - 1 do {
      if (A[j] <= pivot) then {
         i = i + 1;
         swap (A[i], A[j]);
      }
   }
   swap (A[i + 1], A[high]);
   return i + 1;
}
```

Code:

```
#include <bits/stdc++.h>
using namespace std;

void swap(int &a, int &b) {
   int temp = a;
   a = b;
   b = temp;
}
int partition(vector<int> &arr, int low, int high) {
   int pivot = arr[high];
   int i = low - 1;

   for (int j = low; j < high; ++j) {
      if (arr[j] <= pivot) {
         ++i;
         swap(arr[i], arr[j]);
      }
   }
   swap(arr[i + 1], arr[high]);
   return i + 1;
}

int randomizedPartition(vector<int> &arr, int low, int high) {
   int pivotIndex = low + rand() % (high - low + 1);
   swap(arr[pivotIndex], arr[high]);
   return partition(arr, low, high);
```

```
}

void randomizedQuickSort(vector<int> &arr, int low, int high) {
    if (low < high) {
        int pivotIndex = randomizedPartition(arr, low, high);
        randomizedQuickSort(arr, low, pivotIndex - 1);
        randomizedQuickSort(arr, pivotIndex + 1, high);
    }
}

int main() {
    srand(time(0));
    vector<int> arr = {10, 7, 8, 9, 1, 5};
    cout << "Original array: ";
    for (int num : arr) {
        cout << num << " ";
    } cout << endl;

    randomizedQuickSort(arr, 0, arr.size() - 1);

    cout << "Sorted array: ";
    for (int num : arr) {
        cout << num << " ";
    } cout << endl;

    return 0;
}
```

Algorithm Insertion:

```
Algorithm Insert(heap, element){
   heap.push(element);
   index := heap.size() - 1;
   while (index > 0) do {
      parent := (index - 1) / 2;
      if (heap[parent] < heap[index]) then {
         swap(heap[parent], heap[index]);
         index := parent;
      }
      else break;
   }
}
```

Algorithm for Deletion:

```
Algorithm DeleteRoot(heap){
   if (heap.size() = 0) return;
   heap[0] := heap[heap.size() - 1];
   heap.pop_back();
   index := 0;
   while (index < heap.size()) do {
      left := 2 * index + 1;
      right := 2 * index + 2;
      largest := index;
```

```
        if (left < heap.size() and heap[left] > heap[largest]) then
            largest := left;
        if (right < heap.size() and heap[right] > heap[largest]) then
            largest := right;

        if (largest != index) then{
            swap(heap[index], heap[largest]);
            index := largest;
        }
        else break;
    }
}
```

Code:

```
#include <bits/stdc++.h>
using namespace std;

void heapifyUp(vector<int> &heap, int index) {
    while (index > 0) {
        int parent = (index - 1) / 2;
        if (heap[parent] < heap[index]) {
            swap(heap[parent], heap[index]);
            index = parent;
        } else {
            break;
        }
    }
}

void insert(vector<int> &heap, int value) {
    heap.push_back(value);
    heapifyUp(heap, heap.size() - 1);
}

void heapifyDown(vector<int> &heap, int index) {
    int size = heap.size();
    while (index < size) {
        int left = 2 * index + 1;
        int right = 2 * index + 2;
        int largest = index;
```

```
        if (left < size && heap[left] > heap[largest]) {
            largest = left;
        }
        if (right < size && heap[right] > heap[largest]) {
            largest = right;
        }

        if (largest != index) {
            swap(heap[index], heap[largest]);
            index = largest;
        } else {
            break;
        }
    }
}

void deleteRoot(vector<int> &heap) {
    if (heap.empty()) return;

    heap[0] = heap.back();
    heap.pop_back();
    heapifyDown(heap, 0);
}

void printHeap(const vector<int> &heap) {
    for (int num : heap) {
```

```
        cout << num << " ";
    }
    cout << endl;
}

int main() {
    vector<int> heap;

    insert(heap, 10);
    insert(heap, 20);
    insert(heap, 15);
    insert(heap, 30);
    insert(heap, 40);

    cout << "Heap after insertions: ";
    printHeap(heap);

    deleteRoot(heap);
    cout << "Heap after deleting root: ";
    printHeap(heap);

    return 0;
}
```

Algorithm:

```
Algorithm HeapSort(array) {
   BuildMaxHeap(array);
   for i := array.length - 1 down to 1 do {
      swap(array[0], array[i]);
      heapSize := heapSize - 1;
      MaxHeapify(array, 0, heapSize);
   }
}
Algorithm BuildMaxHeap(array) {
   heapSize := array.length;
   for i := (heapSize / 2) - 1 down to 0 do
      MaxHeapify(array, i, heapSize);
}
Algorithm MaxHeapify(array, i, heapSize){
   left := 2 * i + 1;
   right := 2 * i + 2;
   largest := i;
   if (left < heapSize and array[left] > array[largest]) then
      largest := left;
   if (right < heapSize and array[right] > array[largest]) then
      largest := right;
   if (largest != i) then {
      swap(array[i], array[largest]);
      MaxHeapify(array, largest, heapSize);
   }
}
```

Code:

```
#include <bits/stdc++.h>
using namespace std;

void swap(int &a, int &b) {
   int temp = a;
   a = b;
   b = temp;
}

void maxHeapify(vector<int> &arr, int i, int heapSize) {
   int largest = i;
   int left = 2 * i + 1;
   int right = 2 * i + 2;

   if (left < heapSize && arr[left] > arr[largest])
      largest = left;
   if (right < heapSize && arr[right] > arr[largest])
      largest = right;

   if (largest != i) {
      swap(arr[i], arr[largest]);
      maxHeapify(arr, largest, heapSize);
   }
}

void buildMaxHeap(vector<int> &arr) {
   int heapSize = arr.size();
   for (int i = heapSize / 2 - 1; i >= 0; --i) {
```

```
        maxHeapify(arr, i, heapSize);
    }
}

void heapSort(vector<int> &arr) {
    buildMaxHeap(arr);
    for (int i = arr.size() - 1; i > 0; --i) {
        swap(arr[0], arr[i]);
        maxHeapify(arr, 0, i);
    }
}

int main() {
    vector<int> arr = {12, 11, 13, 5, 6, 7};

    cout << "Original array: ";
    for (int num : arr) {
        cout << num << " ";
    } cout << endl;

    heapSort(arr);

    cout << "Sorted array: ";
    for (int num : arr) {
        cout << num << " ";
    } cout << endl;

    return 0;
}
```

Algorithm:

```
Algorithm Initialize(n) {
   for i = 0 to n - 1 do {
      parent[i] := i;
      rank[i] := 0;
   }
}
Algorithm Find(x) {
   if (parent[x] != x) then {
      parent[x] := Find(parent[x]);
   }
   return parent[x];
}
Algorithm Union(x, y) {
   rootX := Find(x);
   rootY := Find(y);
   if (rootX != rootY) then {
      if (rank[rootX] > rank[rootY]) then {
         parent[rootY] := rootX;
      } else if (rank[rootX] < rank[rootY]) then {
         parent[rootX] := rootY;
      } else {
         parent[rootY] := rootX;
         rank[rootX] := rank[rootX] + 1;
      }
   }
}
```

Code:

```
#include <bits/stdc++.h>
using namespace std;

vector<int> parent;
vector<int> rank;

void initialize(int n) {
   parent.resize(n);
   rank.resize(n, 0);
   for (int i = 0; i < n; ++i) {
      parent[i] = i;
   }
}

int find(int x) {
   if (parent[x] != x) {
      parent[x] = find(parent[x]);
   }
   return parent[x];
}

void unionSets(int x, int y) {
   int rootX = find(x);
   int rootY = find(y);

   if (rootX != rootY) {
```

```
        if (rank[rootX] > rank[rootY]) {
            parent[rootY] = rootX;
        } else if (rank[rootX] < rank[rootY]) {
            parent[rootX] = rootY;
        } else {
            parent[rootY] = rootX;
            rank[rootX]++;
        }
    }
}

int main() {
    int n = 5;
    initialize(n);

    unionSets(0, 1);
    unionSets(1, 2);
    unionSets(3, 4);

    cout << "Find(0): " << find(0) << endl;
    cout << "Find(1): " << find(1) << endl;
    cout << "Find(2): " << find(2) << endl;
    cout << "Find(3): " << find(3) << endl;
    cout << "Find(4): " << find(4) << endl;

    return 0;
}
```