**Bài tập cơ sở 2:**

#include <iostream>

using namespace std;

//Chia mảng

int Partition(int a[], int low, int high) {

    int pivot = a[high];

    int i = (low - 1);

    for (int j = low; j < high; j++) {

        if (a[j] <= pivot) {

            i++;

            swap(a[i], a[j]);

        }

    }

    swap(a[i + 1], a[high]);

    return (i + 1);

}

//Sử dụng thuật toán Quick Sort

void QuickSort(int a[], int low, int high) {

    if (low < high) {

        int pi = Partition(a, low, high);

        QuickSort(a, low, pi - 1);

        QuickSort(a, pi + 1, high);

    }

}

//Chia và hợp nhất từng mảng

void Merge(int a[], int l, int m, int r) {

    int n1 = m - l + 1;

    int n2 = r - m;

    int L[n1], R[n2];

    for (int i = 0; i < n1; i++)

        L[i] = a[l + i];

    for (int j = 0; j < n2; j++)

        R[j] = a[m + 1 + j];

    int i = 0, j = 0, k = l;

    while (i < n1 && j < n2) {

        if (L[i] <= R[j]) {

            a[k] = L[i];

            i++;

        } else {

            a[k] = R[j];

            j++;

        }

        k++;

    }

    while (i < n1) {

        a[k] = L[i];

        i++;

        k++;

    }

    while (j < n2) {

        a[k] = R[j];

        j++;

        k++;

    }

}

//Sử dụng thuật toán Merge Sort

void MergeSort(int a[], int l, int r) {

    if (l < r) {

        int m = l + (r - l) / 2;

        MergeSort(a, l, m);

        MergeSort(a, m + 1, r);

        Merge(a, l, m, r);

    }

}

//Xây dựng Heap cho các mảng

void Heapify(int a[], int n, int i) {

    int largest = i;

    int left = 2 \* i + 1;

    int right = 2 \* i + 2;

    if (left < n && a[left] > a[largest])

        largest = left;

    if (right < n && a[right] > a[largest])

        largest = right;

    if (largest != i) {

        swap(a[i], a[largest]);

        Heapify(a, n, largest);

    }

}

//Sử dụng thuật toán Heap Sort

void HeapSort(int a[], int n) {

    for (int i = n / 2 - 1; i >= 0; i--)

        Heapify(a, n, i);

    for (int i = n - 1; i >= 0; i--) {

        swap(a[0], a[i]);

        Heapify(a, i, 0);

    }

}

//In mảng

void PrintA(int a[], int size) {

    for (int i = 0; i < size; i++) {

        cout << a[i] << " ";

    }

    cout << endl;

}

int main() {

    int a1[] = {8, 5, 1, 3, 6, 9, 12, 4, 7, 10};

    int a2[] = {8, 5, 1, 3, 6, 9, 12, 4, 7, 10};

    int a3[] = {8, 5, 1, 3, 6, 9, 12, 4, 7, 10};

    int n = sizeof(a1) / sizeof(a1[0]);

    int QuickA[n];

    copy(a1, a1 + n, QuickA);

    cout << "Quick Sort: ";

    QuickSort(QuickA, 0, n - 1);

    PrintA(QuickA, n);

    int MergeA[n];

    copy(a2, a2 + n, MergeA);

    cout << "Merge Sort: ";

    MergeSort(MergeA, 0, n - 1);

    PrintA(MergeA, n);

    int HeapA[n];

    copy(a3, a3 + n, HeapA);

    cout << "Heap Sort: ";

    HeapSort(HeapA, n);

    PrintA(HeapA, n);

    return 0;

}

* **Quick Sort:**

Ý niệm: Chia mảng ra thành 2 phần dựa trên pivot sau đó gọi đệ quy lên 2 mảng để tiến hành sắp xếp mảng.

Vd với mảng {8, 5, 1, 3, 6, 9, 12, 4, 7, 10}:

1. Chọn 6 làm pivot: Mảng trước 6 gồm {8, 5, 1, 3} và sau 6 gồm {9, 12, 4, 7, 10}.
2. Tiến hành so sánh, đệ quy và sắp xếp các phần tử trong mảng bắt đầu từ các phần tử ngoài cùng: 8 < 10 nhưng 8 > 6 nên sẽ được sắp xếp thành {6, 8, 10}.
3. Chương trình sẽ được tiếp tục cho tới khi cho ra kết quả là {1, 3, 4, 5, 6, 7, 8, 9, 10, 12}.

* **Merge Sort:**

Ý niệm: Chia mảng ra và rồi hợp nhất từng phần tử trong mảng sau đó gọi đệ quy để tiến hành sắp xếp mảng.

Vd với mảng {8, 5, 1, 3, 6, 9, 12, 4, 7, 10}:

1. Chia nhỏ mảng ra thành từng phần tử.
2. Tiến hành so sánh, đệ quy, sắp xếp 2 phần tử với nhau: 8 > 5 nên sẽ được sắp xếp thành {5, 8}.
3. Chương trình sẽ được tiếp tục cho tới khi cho ra kết quả là {1, 3, 4, 5, 6, 7, 8, 9, 10, 12}.

* **Heap Sort:**

Ý niệm: Xây dựng cây heap và rồi chọn phần tử lớn nhất trong mảng làm phần tử gốc sau đó thực hiện “heapify” để tiến hành sắp xếp mảng.

Vd với mảng {8, 5, 1, 3, 6, 9, 12, 4, 7, 10}:

1. Chọn 12 làm phần tử gốc và hoán đổi với phần tử cuối cùng của mảng.
2. Tiến hành so sánh và sắp xếp các phần tử còn lại trong mảng bằng “heapify”.
3. Chương trình sẽ được tiếp tục cho tới khi cho ra kết quả là {1, 3, 4, 5, 6, 7, 8, 9, 10, 12}.