**1.Max heapify-Heapsort with Linked List-(ascending order)**

|  |  |
| --- | --- |
| #include <iostream>  using namespace std;  class Node {  public:  int val;  Node \*next;  Node(int val) {  this->val = val;  this->next = nullptr;  }  };  class SortByValueComparator {  public:  int compare(Node \*node1, Node \*node2) {  if (node1->val < node2->val) {  return -1;  } else if (node1->val > node2->val) {  return 1;  }  return 0;  }  };  SortByValueComparator sortByValueComparator;  void heapify(Node \*\*arr, int n, int i) {  int largest = i;  int right = 2 \* i + 2;  int left = 2 \* i + 1;  if (left < n && sortByValueComparator.compare(arr[left], arr[largest]) > 0) {  largest = left;  }  if (right < n && sortByValueComparator.compare(arr[right], arr[largest]) > 0) {  largest = right;  }  if (largest != i) {  swap(arr[i]->val, arr[largest]->val);  heapify(arr, n, largest);  }  }  void sortArray(Node \*\*arr, int n) {  for (int i = n / 2 - 1; i >= 0; i--) {  heapify(arr, n, i);  }  for (int i = n - 1; i > 0; i--) {  swap(arr[0]->val, arr[i]->val);  heapify(arr, i, 0);  }  int v;  cin >> v;  insert\_at\_tail(head, v);  }  print\_linked\_list(head);  cout << endl;  heapsort(head);  return 0;  } | }  void heapsort(Node \*head) {  Node \*current = head;  int size = 0;  while (current != nullptr) {  size++;  current = current->next;  }  Node \*\*arr = new Node \*[size];  current = head;  int i = 0;  while (current != nullptr) {  arr[i++] = current;  current = current->next;  }  sortArray(arr, size);  for (int i = 0; i < size; i++) {  cout << arr[i]->val << " ";  }  cout << endl;  delete[] arr;  }  void insert\_at\_tail(Node \*&head, int v) {  Node \*newNode = new Node(v);  if (head == nullptr) {  head = newNode;  return;  }  Node \*tmp = head;  while (tmp->next != nullptr) {  tmp = tmp->next;  }  tmp->next = newNode;  }  void print\_linked\_list(Node \*head) {  Node \*tmp = head;  while (tmp != nullptr) {  cout << tmp->val << " ";  tmp = tmp->next;  }  }  int main() {  Node \*head = nullptr;  int n;    cin >> n;    for (int i = 0; i < n; i++) { |

**2. Max heapify-Heapsort with Linked List-(descending order)**

|  |  |
| --- | --- |
| #include <iostream>  using namespace std;  class Node {  public:  int val;  Node \*next;  Node(int val) {  this->val = val;  this->next = nullptr;  }  };  class SortByValueComparator {  public:  int compare(Node \*node1, Node \*node2) {  if (node1->val < node2->val) {  return -1;  } else if (node1->val > node2->val) {  return 1;  }  return 0;  }  };  SortByValueComparator sortByValueComparator;  void heapify(Node \*\*arr, int n, int i) {  int smallest = i;  int right = 2 \* i + 2;  int left = 2 \* i + 1;  if (left < n && sortByValueComparator.compare(arr[left], arr[smallest]) < 0) {  smallest = left;  }  sortArray(arr, size);  for (int i = 0; i < size; i++) {  cout << arr[i]->val << " ";  }  cout << endl;  delete[] arr;  }  void insert\_at\_tail(Node \*&head, int v) {  Node \*newNode = new Node(v);  if (head == nullptr) {  head = newNode;  return;  }  Node \*tmp = head;  while (tmp->next != nullptr) {  tmp = tmp->next;  }  tmp->next = newNode;  } | if (right < n && sortByValueComparator.compare(arr[right], arr[smallest]) < 0) {  smallest = right;  }  if (smallest != i) {  swap(arr[i]->val, arr[smallest]->val);  heapify(arr, n, smallest);  }  }  void sortArray(Node \*\*arr, int n) {  for (int i = n / 2 - 1; i >= 0; i--) {  heapify(arr, n, i);  }  for (int i = n - 1; i > 0; i--) {  swap(arr[0]->val, arr[i]->val);  heapify(arr, i, 0);  }  }  void heapsort(Node \*head) {  Node \*current = head;  int size = 0;  while (current != nullptr) {  size++;  current = current->next;  }  Node \*\*arr = new Node \*[size];  current = head;  int i = 0;  while (current != nullptr) {  arr[i++] = current;  current = current->next;  }  void print\_linked\_list(Node \*head) {  Node \*tmp = head;  while (tmp != nullptr) {  cout << tmp->val << " ";  tmp = tmp->next;  }  }  int main() {  Node \*head = nullptr;  int n;  cin >> n;  for (int i = 0; i < n; i++) {  int v;  cin >> v;  insert\_at\_tail(head, v);  }  print\_linked\_list(head);  cout << endl;  heapsort(head);  return 0;  } |

**3. Insertion Sort with Linked List.**

|  |  |
| --- | --- |
| #include <iostream>  using namespace std;  class Node {  public:  int val;  Node \*next;  Node(int val) {  this->val = val;  this->next = nullptr;  }  };  void insert\_at\_tail(Node \*&head, int v) {  Node \*newNode = new Node(v);  if (head == nullptr) {  head = newNode;  return;  } | Node \*tmp = head;  while (tmp->next != nullptr) {  tmp = tmp->next;  }  tmp->next = newNode;  }  void print\_linked\_list(Node \*head) {  Node \*tmp = head;  while (tmp != nullptr) {  cout << tmp->val << " ";  tmp = tmp->next;  }  }  void insertionSort(Node\*& head) {  if (head == nullptr || head->next == nullptr)  return;  Node\* sorted = nullptr;  Node\* current = head;  while (current != nullptr) {  Node\* nextNode = current->next; |

|  |  |
| --- | --- |
| ode\* nextNode = current->next;  if (sorted == nullptr || sorted->val >= current->val) {  current->next = sorted;  sorted = current;  } else {  Node\* temp = sorted;  while (temp->next != nullptr && temp->next->val < current->val) {  temp = temp->next;  }  current->next = temp->next;  temp->next = current;  }  current = nextNode;  }  head = sorted;  } | int main() {  Node \*head = nullptr;  int n;  cin >> n;  for (int i = 0; i < n; i++) {  int v;  cin >> v;  insert\_at\_tail(head, v);  }  print\_linked\_list(head);  cout<<endl;  insertionSort(head);  print\_linked\_list(head);  } |