|  |  |
| --- | --- |
| **Extra Practical No: 12 & 13** | |
|  | |
| **Explanation/ Stepwise-Procedure/ Algorithm** | Write a program to sort arrays using Radix sort Create a Heap Tree. Perform the following operations: insert node, delete node in Heap |
| **Theory:** | Write the following questions  * What is a Radix sort   Radix Sort is a non-comparative sorting algorithm that sorts numbers by processing individual digits. It sorts integers by grouping numbers based on each digit's place value, such as units, tens, hundreds, and so on. Radix Sort processes these place values one at a time, sorting numbers from the least significant digit (LSD) to the most significant digit (MSD) in each pass.   * Discuss time complexity of radix sort.   **Best case**: O(d×(n+k))  **Average case**: O(d×(n+k))  **Worst case**: O(d×(n+k))   * What is heap tree.   A heap is a complete binary tree, meaning all levels are fully filled except possibly the last level, which is filled from left to right. |
| **Source Code/Algorithm/Flow Chart:** | Radix sort  #include <stdio.h>  #include <stdlib.h>  int getMax(int arr[], int n) {  int max = arr[0];  for (int i = 1; i < n; i++)  if (arr[i] > max)  max = arr[i];  return max;  }  void countSort(int arr[], int n, int exp) {  int output[n];  int count[10] = {0};  for (int i = 0; i < n; i++)  count[(arr[i] / exp) % 10]++;  for (int i = 1; i < 10; i++)  count[i] += count[i - 1];  for (int i = n - 1; i >= 0; i--) {  output[count[(arr[i] / exp) % 10] - 1] = arr[i];  count[(arr[i] / exp) % 10]--;  }  for (int i = 0; i < n; i++)  arr[i] = output[i];  }  void radixSort(int arr[], int n) {  int max = getMax(arr, n);  for (int exp = 1; max / exp > 0; exp \*= 10)  countSort(arr, n, exp);  }  int main() {  int arr[] = {170, 45, 75, 90, 802, 24, 2, 66};  int n = sizeof(arr) / sizeof(arr[0]);  radixSort(arr, n);  printf("Sorted array: ");  for (int i = 0; i < n; i++)  printf("%d ", arr[i]);  return 0;  }  Heap tree  #include <stdio.h>  #include <stdlib.h>  #define MAX\_SIZE 100  typedef struct {  int data[MAX\_SIZE];  int size;  } MinHeap;  void heapifyUp(MinHeap \*heap, int index) {  while (index > 0 && heap->data[(index - 1) / 2] > heap->data[index]) {  int temp = heap->data[index];  heap->data[index] = heap->data[(index - 1) / 2];  heap->data[(index - 1) / 2] = temp;  index = (index - 1) / 2;  }  }  void heapifyDown(MinHeap \*heap, int index) {  int smallest = index;  int left = 2 \* index + 1;  int right = 2 \* index + 2;  if (left < heap->size && heap->data[left] < heap->data[smallest])  smallest = left;  if (right < heap->size && heap->data[right] < heap->data[smallest])  smallest = right;  if (smallest != index) {  int temp = heap->data[index];  heap->data[index] = heap->data[smallest];  heap->data[smallest] = temp;  heapifyDown(heap, smallest);  }  }  void insertNode(MinHeap \*heap, int value) {  if (heap->size == MAX\_SIZE) {  printf("Heap is full!\n");  return;  }  heap->data[heap->size] = value;  heap->size++;  heapifyUp(heap, heap->size - 1);  }  void deleteMin(MinHeap \*heap) {  if (heap->size == 0) {  printf("Heap is empty!\n");  return;  }  heap->data[0] = heap->data[heap->size - 1];  heap->size--;  heapifyDown(heap, 0);  }  void displayHeap(MinHeap \*heap) {  for (int i = 0; i < heap->size; i++)  printf("%d ", heap->data[i]);  printf("\n");  }  int main() {  MinHeap heap;  heap.size = 0;  insertNode(&heap, 10);  insertNode(&heap, 15);  insertNode(&heap, 20);  insertNode(&heap, 17);  insertNode(&heap, 8);  printf("Heap after insertions: ");  displayHeap(&heap);  deleteMin(&heap);  printf("Heap after deleting the root node: ");  displayHeap(&heap);  return 0;  } |
| **Output Screenshots (if applicable)** | Radix sort |
|  | Heap tree |
| **Conclusion** | Thus, we have studied and implemented the Threaded Binary Tree and traversals on it. |
| **Post Lab Questions:** | * What are applications of radix sort |