1. **Merge Sort**

#include <stdio.h>

void merge(int arr[], int start, int mid, int end) {

int i = start, j = mid + 1, k = 0;

int temp[100];

while (i <= mid && j <= end) {

if (arr[i] < arr[j]) {

temp[k++] = arr[i++];

} else {

temp[k++] = arr[j++];

}

}

while (i <= mid) temp[k++] = arr[i++];

while (j <= end) temp[k++] = arr[j++];

for (i = start, k = 0; i <= end; i++, k++) {

arr[i] = temp[k];

}

}

void mergeSort(int arr[], int start, int end) {

if (start < end) {

int mid = (start + end) / 2;

mergeSort(arr, start, mid);

mergeSort(arr, mid + 1, end);

merge(arr, start, mid, end);

}

}

void printArray(int arr[], int size) {

for (int i = 0; i < size; i++) printf("%d ", arr[i]);

printf("\n");

}

int main() {

int arr[] = {5, 2, 9, 1, 6};

int n = 5;

printf("Before sorting: ");

printArray(arr, n);

mergeSort(arr, 0, n - 1);

printf("After Merge Sort: ");

printArray(arr, n);

return 0;

}

T(n) = 2T(n/2) + O(n)

By solving this recurrence (Master Theorem), we get:

**Time Complexity:**

* **Best Case:** O(n log n)
* **Average Case:** O(n log n)
* **Worst Case:** O(n log n)

1. **Bucket Sort**

#include <stdio.h>

void insertionSort(int arr[], int n) {

for (int i = 1; i < n; i++) {

int key = arr[i];

int j = i - 1;

while (j >= 0 && arr[j] > key) {

arr[j + 1] = arr[j];

j--;

}

arr[j + 1] = key;

}

}

void bucketSort(int arr[], int n) {

int buckets[10][10];

int bucketCount[10] = {0};

for (int i = 0; i < n; i++) {

int index = arr[i] / 10;

buckets[index][bucketCount[index]++] = arr[i];

}

for (int i = 0; i < 10; i++) {

if (bucketCount[i] > 0)

insertionSort(buckets[i], bucketCount[i]);

}

int k = 0;

for (int i = 0; i < 10; i++) {

for (int j = 0; j < bucketCount[i]; j++) {

arr[k++] = buckets[i][j];

}

}

}

void printArray(int arr[], int n) {

for (int i = 0; i < n; i++) printf("%d ", arr[i]);

printf("\n");

}

int main() {

int arr[] = {29, 25, 3, 49, 9, 37, 21, 43};

int n = 8;

printf("Before sorting: ");

printArray(arr, n);

bucketSort(arr, n);

printf("After Bucket Sort: ");

printArray(arr, n);

return 0;

}

**Best Case**

* Each bucket has roughly n/k elements
* Insertion Sort on each bucket → O((n/k)²)
* Total cost:

O(n + k) + k \* O((n/k)²) = O(n + n²/k)

If k = n, then:

O(n + n²/n) = O(n + n) = O(n)

**Average Case:**

* Elements are randomly distributed → each bucket has ≈ n/k elements
* Sorting each bucket takes O((n/k)²), and there are k buckets:

O(n + k) + k \* O((n/k)²) = O(n + n²/k)

**Worst Case:**

* All elements go into one bucket → O(n²)

1. **Quick Sort**

#include <stdio.h>

void swap(int \*a, int \*b) {

int temp = \*a;

\*a = \*b;

\*b = temp;

}

int partition(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;

}

// Quick Sort function

void quickSort(int arr[], int low, int high) {

if (low < high) {

int pivotIndex = partition(arr, low, high);

quickSort(arr, low, pivotIndex - 1);

quickSort(arr, pivotIndex + 1, high);

}

}

void printArray(int arr[], int n) {

for (int i = 0; i < n; i++) printf("%d ", arr[i]);

printf("\n");

}

int main() {

int arr[] = {10, 7, 8, 9, 1, 5};

int n = 6;

printf("Before sorting: ");

printArray(arr, n);

quickSort(arr, 0, n - 1);

printf("After Quick Sort: ");

printArray(arr, n);

return 0;

}

**Average Case:**

* Partitioning splits the array into two balanced parts (say n/2)
* Each level of recursion does O(n) work to partition

T(n) = 2T(n/2) + O(n)

Same as Merge Sort → O(n log n)

**Best Case:**

* Pivot always splits array evenly
* Same as average case: **O(n log n)**

**Worst Case:**

* Pivot is smallest or largest element → one part has size 0, other has n - 1

T(n) = T(n-1) + O(n)

T(n) = n + (n - 1) + (n - 2) + ... + 1 = O(n²)