**Implementation of QuickSort and Randomized QuickSort with comparative analysis.**

Theory:

QuickSort is a[Divide and Conquer algorithm](https://www.geeksforgeeks.org/divide-and-conquer-algorithm-introduction/). It picks an element as pivot and partitions the given array around the picked pivot. There are two partition schemes in quick sort:

* Lomuto’s Partition Scheme: This algorithm works by assuming the pivot element as the last element. If any other element is given as a pivot element then swap it first with the last element. Now initialize two variables i as low and j also low,  iterate over the array and increment i when arr[j] <= pivot and swap arr[i] with arr[j] otherwise increment only i. After coming out from the loop swap arr[i] with arr[hi]. This i stores the pivot element.
* Hoare’s Partition Scheme: [Hoare’s Partition Scheme](https://en.wikipedia.org/wiki/Quicksort#Hoare_partition_scheme) works by initializing two indexes that start at two ends, the two indexes move toward each other until an inversion is (A smaller value on the left side and greater value on the right side) found. When an inversion is found, two values are swapped and the process is repeated.

Code:

Normal Hoare Partition

| #include <stdio.h> #include <iostream> #include <time.h> #include <chrono>  using namespace std; using namespace std::chrono;  int partition(int arr[], int low, int high) {     int pivot = arr[low];     int i = low - 1, j = high + 1;      while (true)     {         do         {             i++;         } while (arr[i] < pivot);          do         {             j--;         } while (arr[j] > pivot);          if (i >= j)             return j;          swap(arr[i], arr[j]);     } }  void quickSort(int arr[], int low, int high) {     if (low < high)     {         int pi = partition(arr, low, high);         quickSort(arr, low, pi);         quickSort(arr, pi + 1, high);     } }  void printArray(int arr[], int size) {     int i;     for (i = 0; i < size; i++)         cout << arr[i] << ' ';     cout << endl; }  int main() {     int arr[] = {99, 81, 56, 63, 17, 6, 84, 87, 62, 5, 7, 38, 67, 22, 10, 37, 90, 85, 25, 1, 88, 42, 16, 2, 91, 95, 78, 50, 97, 82, 52, 69, 32, 43, 40, 53, 73, 61, 30, 26, 33, 70, 74, 35, 31, 65, 11, 24, 27, 54, 57, 15, 66, 98, 44, 19, 49, 13, 60, 39, 8, 80, 3, 86, 18, 34, 59, 72, 93, 71, 21, 4, 23, 64, 9, 29, 47, 55, 46, 89, 79, 51, 100, 36, 75, 58, 68, 41, 94, 28, 83, 20, 48, 45, 77, 14, 12, 96, 76, 92};     int n = sizeof(arr) / sizeof(arr[0]);     cout << "Unsorted Array:" << endl;     printArray(arr, n);     auto start = high\_resolution\_clock::now();     quickSort(arr, 0, n - 1);     auto stop = high\_resolution\_clock::now();     cout << "Sorted Array:" << endl;     printArray(arr, n);     auto duration = duration\_cast<microseconds>(stop - start);     cout << "Time taken by function: " << duration.count() << " microseconds" << endl;     return 0; } |
| --- |

Normal Lomuto Partition

| #include <stdio.h> #include <iostream> #include <time.h> #include <chrono>  using namespace std; using namespace std::chrono;  int partition(int arr[], int low, int high) {     int pivot = arr[high];     int i = (low - 1);      for (int j = low; j <= high - 1; j++)     {         if (arr[j] <= pivot)         {             i++;             swap(arr[i], arr[j]);         }     }     swap(arr[i + 1], arr[high]);     return (i + 1); }  void quickSort(int arr[], int low, int high) {     if (low < high)     {         int pi = partition(arr, low, high);         quickSort(arr, low, pi - 1);         quickSort(arr, pi + 1, high);     } } void printArray(int arr[], int size) {     int i;     for (i = 0; i < size; i++)         cout << arr[i] << " ";     cout << endl; }  int main() {     int arr[] = {99, 81, 56, 63, 17, 6, 84, 87, 62, 5, 7, 38, 67, 22, 10, 37, 90, 85, 25, 1, 88, 42, 16, 2, 91, 95, 78, 50, 97, 82, 52, 69, 32, 43, 40, 53, 73, 61, 30, 26, 33, 70, 74, 35, 31, 65, 11, 24, 27, 54, 57, 15, 66, 98, 44, 19, 49, 13, 60, 39, 8, 80, 3, 86, 18, 34, 59, 72, 93, 71, 21, 4, 23, 64, 9, 29, 47, 55, 46, 89, 79, 51, 100, 36, 75, 58, 68, 41, 94, 28, 83, 20, 48, 45, 77, 14, 12, 96, 76, 92};     int n = sizeof(arr) / sizeof(arr[0]);     cout << "Unsorted Array:" << endl;     printArray(arr, n);     auto start = high\_resolution\_clock::now();     quickSort(arr, 0, n - 1);     auto stop = high\_resolution\_clock::now();     cout << "Sorted Array" << endl;     printArray(arr, n);     auto duration = duration\_cast<microseconds>(stop - start);     cout << "Time taken by function: " << duration.count() << " microseconds" << endl;     return 0; } |
| --- |

Randomized Hoare Partition

| #include <stdio.h> #include <iostream> #include <time.h> #include <chrono> using namespace std; using namespace std::chrono;  int partition(int arr[], int low, int high) {     int pivot = arr[low];     int i = low - 1, j = high + 1;      while (true)     {         do         {             i++;         } while (arr[i] < pivot);          do         {             j--;         } while (arr[j] > pivot);          if (i >= j)             return j;          swap(arr[i], arr[j]);     } }  int random\_partition(int arr[], int low, int high) {     int r = low + rand() % (high - low);     swap(arr[r], arr[low]);     return partition(arr, low, high); }  void quickSort(int arr[], int low, int high) {     if (low < high)     {         int pi = random\_partition(arr, low, high);         quickSort(arr, low, pi);         quickSort(arr, pi + 1, high);     } }  void printArray(int arr[], int size) {     int i;     for (i = 0; i < size; i++)         cout << arr[i] << ' ';     cout << endl; } int main() {     int arr[] = {99, 81, 56, 63, 17, 6, 84, 87, 62, 5, 7, 38, 67, 22, 10, 37, 90, 85, 25, 1, 88, 42, 16, 2, 91, 95, 78, 50, 97, 82, 52, 69, 32, 43, 40, 53, 73, 61, 30, 26, 33, 70, 74, 35, 31, 65, 11, 24, 27, 54, 57, 15, 66, 98, 44, 19, 49, 13, 60, 39, 8, 80, 3, 86, 18, 34, 59, 72, 93, 71, 21, 4, 23, 64, 9, 29, 47, 55, 46, 89, 79, 51, 100, 36, 75, 58, 68, 41, 94, 28, 83, 20, 48, 45, 77, 14, 12, 96, 76, 92};     int n = sizeof(arr) / sizeof(arr[0]);     auto start = high\_resolution\_clock::now();     quickSort(arr, 0, n - 1);     auto stop = high\_resolution\_clock::now();     cout << "Sorted Array" << endl;     printArray(arr, n);     auto duration = duration\_cast<microseconds>(stop - start);     cout << "Time taken by function: " << duration.count() << " microseconds" << endl;     return 0; } |
| --- |

Randomized Lomuto Partition

| #include <stdio.h> #include <iostream> #include <time.h> #include <chrono> using namespace std; using namespace std::chrono; // Lomuto Partitioning int partition(int arr[], int low, int high) {     int pivot = arr[high];     int i = (low - 1);      for (int j = low; j <= high - 1; j++)     {         if (arr[j] <= pivot)         {             i++;             swap(arr[i], arr[j]);         }     }     swap(arr[i + 1], arr[high]);     return (i + 1); } int random\_partition(int arr[], int low, int high) {     int r = low + rand() % (high - low);     swap(arr[r], arr[high]);     return partition(arr, low, high); }  void quickSort(int arr[], int low, int high) {     if (low < high)     {         int pi = random\_partition(arr, low, high);         quickSort(arr, low, pi - 1);         quickSort(arr, pi + 1, high);     } } void printArray(int arr[], int size) {     int i;     for (i = 0; i < size; i++)         cout << arr[i] << ' ';     cout << endl; }  int main() {     int arr[] = {99, 81, 56, 63, 17, 6, 84, 87, 62, 5, 7, 38, 67, 22, 10, 37, 90, 85, 25, 1, 88, 42, 16, 2, 91, 95, 78, 50, 97, 82, 52, 69, 32, 43, 40, 53, 73, 61, 30, 26, 33, 70, 74, 35, 31, 65, 11, 24, 27, 54, 57, 15, 66, 98, 44, 19, 49, 13, 60, 39, 8, 80, 3, 86, 18, 34, 59, 72, 93, 71, 21, 4, 23, 64, 9, 29, 47, 55, 46, 89, 79, 51, 100, 36, 75, 58, 68, 41, 94, 28, 83, 20, 48, 45, 77, 14, 12, 96, 76, 92};     int n = sizeof(arr) / sizeof(arr[0]);     cout << "Unsorted Array" << endl;     printArray(arr, n);     auto start = high\_resolution\_clock::now();     quickSort(arr, 0, n - 1);     auto stop = high\_resolution\_clock::now();     cout << "Sorted Array" << endl;     printArray(arr, n);     auto duration = duration\_cast<microseconds>(stop - start);     cout << "Time taken by function: " << duration.count() << " microseconds" << endl;     return 0; } |
| --- |

Output:

Normal Hoare Partition

| D:\Files\Engineering\Semester - 6\AA\Experiments\Experiment 7>g++ -Wall -std=c++14 normal\_hoare.cpp -o normal\_hoare  D:\Files\Engineering\Semester - 6\AA\Experiments\Experiment 7>normal\_hoare Unsorted Array: 99 81 56 63 17 6 84 87 62 5 7 38 67 22 10 37 90 85 25 1 88 42 16 2 91 95 78 50 97 82 52 69 32 43 40 53 73 61 30 26 33 70 74 35 31 65 11 24 27 54 57 15 66 98 44 19 49 13  60 39 8 80 3 86 18 34 59 72 93 71 21 4 23 64 9 29 47 55 46 89 79 51 100 36 75 58 68 41 94 28 83 20 48 45 77 14 12 96 76 92 Sorted Array: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 Time taken by function: 13 microseconds |
| --- |

Normal Lomuto Partition

| D:\Files\Engineering\Semester - 6\AA\Experiments\Experiment 7>g++ -Wall -std=c++14 normal\_lomuto.cpp -o normal\_lomuto  D:\Files\Engineering\Semester - 6\AA\Experiments\Experiment 7>normal\_lomuto Unsorted Array: 99 81 56 63 17 6 84 87 62 5 7 38 67 22 10 37 90 85 25 1 88 42 16 2 91 95 78 50 97 82 52 69 32 43 40 53 73 61 30 26 33 70 74 35 31 65 11 24 27 54 57 15 66 98 44 19 49 13  60 39 8 80 3 86 18 34 59 72 93 71 21 4 23 64 9 29 47 55 46 89 79 51 100 36 75 58 68 41 94 28 83 20 48 45 77 14 12 96 76 92 Sorted Array 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 Time taken by function: 14 microseconds |
| --- |

Random Hoare Partition

| D:\Files\Engineering\Semester - 6\AA\Experiments\Experiment 7>g++ -Wall -std=c++14 random\_hoare.cpp -o random\_hoare  D:\Files\Engineering\Semester - 6\AA\Experiments\Experiment 7>random\_hoare Unsorted Array 99 81 56 63 17 6 84 87 62 5 7 38 67 22 10 37 90 85 25 1 88 42 16 2 91 95 78 50 97 82 52 69 32 43 40 53 73 61 30 26 33 70 74 35 31 65 11 24 27 54 57 15 66 98 44 19 49 13  60 39 8 80 3 86 18 34 59 72 93 71 21 4 23 64 9 29 47 55 46 89 79 51 100 36 75 58 68 41 94 28 83 20 48 45 77 14 12 96 76 92 Sorted Array 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 Time taken by function: 12 microseconds |
| --- |

Random Lomuto Partition’

| D:\Files\Engineering\Semester - 6\AA\Experiments\Experiment 7>g++ -Wall -std=c++14 random\_lomuto.cpp -o random\_lomuto  D:\Files\Engineering\Semester - 6\AA\Experiments\Experiment 7>random\_lomuto Unsorted Array 99 81 56 63 17 6 84 87 62 5 7 38 67 22 10 37 90 85 25 1 88 42 16 2 91 95 78 50 97 82 52 69 32 43 40 53 73 61 30 26 33 70 74 35 31 65 11 24 27 54 57 15 66 98 44 19 49 13  60 39 8 80 3 86 18 34 59 72 93 71 21 4 23 64 9 29 47 55 46 89 79 51 100 36 75 58 68 41 94 28 83 20 48 45 77 14 12 96 76 92 Sorted Array 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 Time taken by function: 15 microseconds |
| --- |

Observations:

Comparing Hoare partitioning with Lomuto partitioning, Hoare partitioning is more efficient as it takes 13 microseconds while Lomuto partitioning takes 14 microseconds. Randomized Hoare partitioning is even more efficient as it takes 12 microseconds while randomized Lomuto partitioning is less efficient with a runtime of 15 microseconds.

Conclusion:

Thus randomized Hoare partitioning is the most efficient partitioning method amongst all the methods that we implemented. Hoare partitioning is more efficient than Lomuto partitioning because it does three times fewer swaps on an average. It also creates efficient partitions even when all values are equal.

References:

<https://www.geeksforgeeks.org/hoares-vs-lomuto-partition-scheme-quicksort/>