PROJECT REPORT

18CSC204J – DESIGN and ANALYSIS of ALGORITHM

Submitted by

ISHITA SHARMA [RA2011027010086]

And

AAYUSH ANAND [RA2011027010086]

Semester IV

In partial satisfaction of the requirements for the degree of

BACHELOR OF TECHNOLOGY
in
COMPUTER SCIENCE ENGINEERING

with specialization in BIG DATA

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

SRM NAGAR KATTANKULATHUR – 603203,

KANCHEEPURAM DISTRICT

JUNE 2022

PROJECT REPORT

18CSC204J – DESIGN and ANALYSIS of ALGORITHM

Submitted by

ISHITA SHARMA [RA2011027010086]

And

AAYUSH ANAND [RA2011027010086]

Semester IV

In partial satisfaction of the requirements for the degree of

BACHELOR OF TECHNOLOGY
in
COMPUTER SCIENCE ENGINEERING

with specialization in BIG DATA

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

SRM NAGAR KATTANKULATHUR – 603203,

KANCHEEPURAM DISTRICT

JUNE 2022

SRM
INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)

COLLEGE OF ENGINEERING & TECHNOLOGY
SRM INSTITUTE OF SCIENCE & TECHNOLOGY
S.R.M. NAGAR, KATTANKULATHUR - 603203
Chengalpattu District

BONAFIDE CERTIFICATE

RA2011027010075
Register No RA2011027010086

Certified to be the bonafide record of work done by Aayush Anand and Ishita Sharma of Data Science & Business System B.Tech Degree course in the Practical 18CSC204J - DAA in **SRM INSTITUTE OF SCIENCE & TECHNOLOGY,** Kattankulathur during the academic year 2022.

DATE: **LAB INCHARGE**

DATE: **PROFESSOR I/C/ACADEMIC ADVSIOR**

Submitted for the University Practical Examination held on ____________, in ______________________ **SRM INSTITUTE OF SCIENCE & TECHNOLOGY,** Kattankulathur.

Internal Examiner 1 **Internal Examiner 2**

Name : Aayush Anand and Ishita Sharma Class : W-1 section

Reg. No.: RA2011027010075 and RA2011027010086 Branch : Big Data Analytics (CSE)

INDEX

ABSTRACT

An algorithm is any well-defined procedure or set of instructions. The goal of this research is to perform and intensive empirical analysis of the new developed rule and gift its practicality.

The results of the analysis proved that a hybrid merge-heap sorting (meap sort) algorithm is much more efficient then the other algorithms having $O(n^2)$ time complexity like bubble, selection, insertion sort and even the normal merge sort.

This paper also aims to give out a detailed comparison of the hybrid merge heap algorithm with merge sort, bubble sort and other conventional sorting methods.

• KEYWORDS :
→ Merge-heap sort (Meap sort)
→ Algorithms
→ Selection sort
→ Bubble sort
→ Insertion sort
→ Time Complexity
→ Asymptotic notations

INTRODUCTION

The proposed algorithm is mainly based on two algorithms, namely:

Merge Sort and Max-Heap Sort

It uses the 'divide and Conquer' approach as its backbone. The algorithm works in 3 steps

→ The unsorted array is divided into groups of three elements, using the concept of merge sort.

→ Each of the groups of three elements are sorted using Max-Heap sort. That is, now we have several sorted groups of three elements each.

→ Now merge sort is applied once again. The groups of three are joined to make larger groups stepwise and sorting is done. The process continues untill the merging gives a final sorted array.

TECHNIQUE USED – DIVIDE AND CONQUER

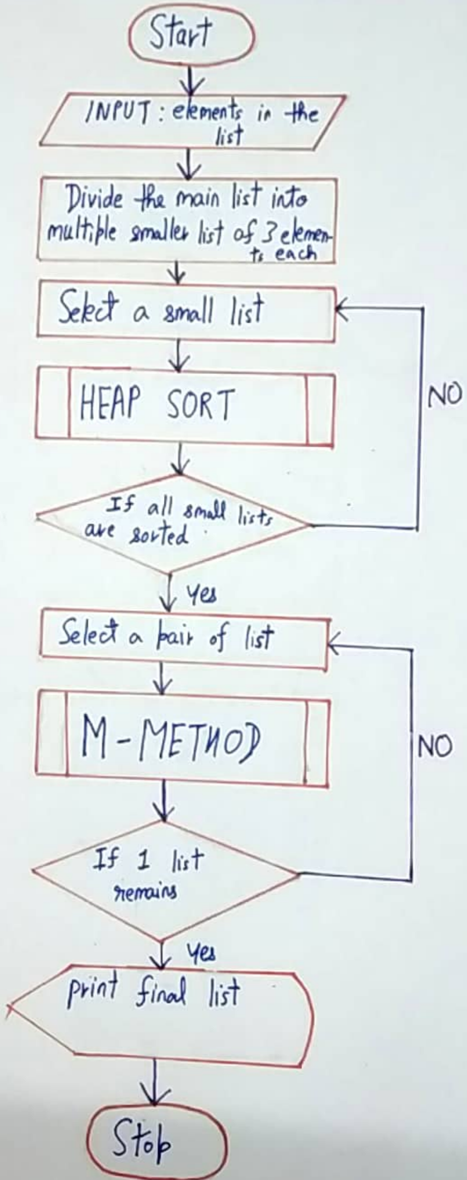
The proposed algorithm is based on Merge and Heap sort, and uses the 'divide and conquer' approach as a backbone.

Divide and Conquer approach works recursively breaking down of a problem into two or more subproblems of same or related type, until these become simple enough to be solved directly. The solutions to the sub-problems are then combined to give a solution to the original problem.

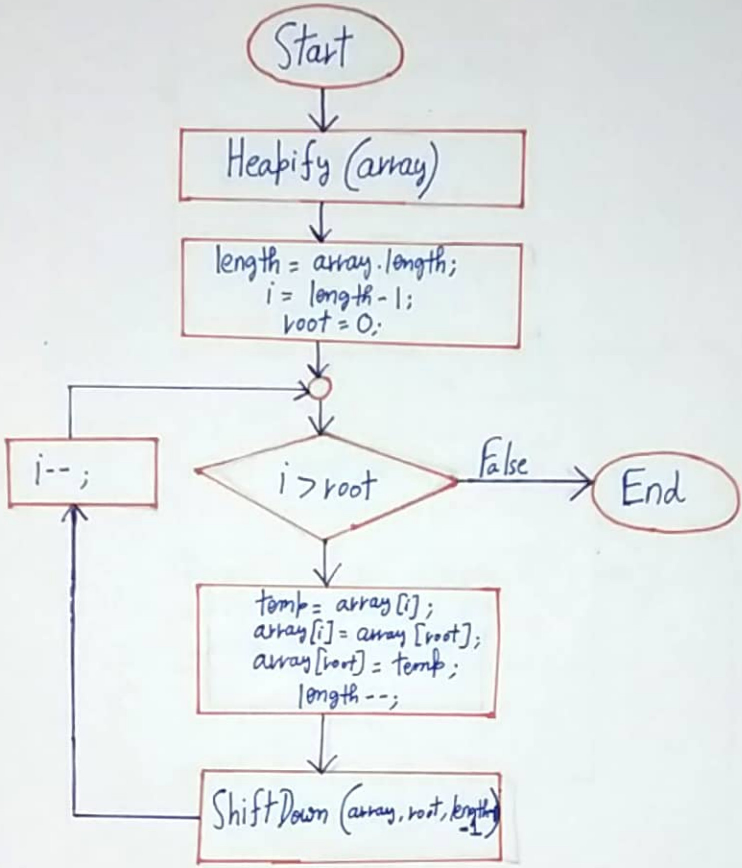
The Divide and Conquer paradigm is often used to find the optimal solution of a problem, and this technique is the basis for efficient sorting algorithms like Quick sort and Merge sort, finding the closest pair of points, multiplying large numbers.

ALGORITHM

⊙FLOWCHART- DISPLAYING MEAP SORT

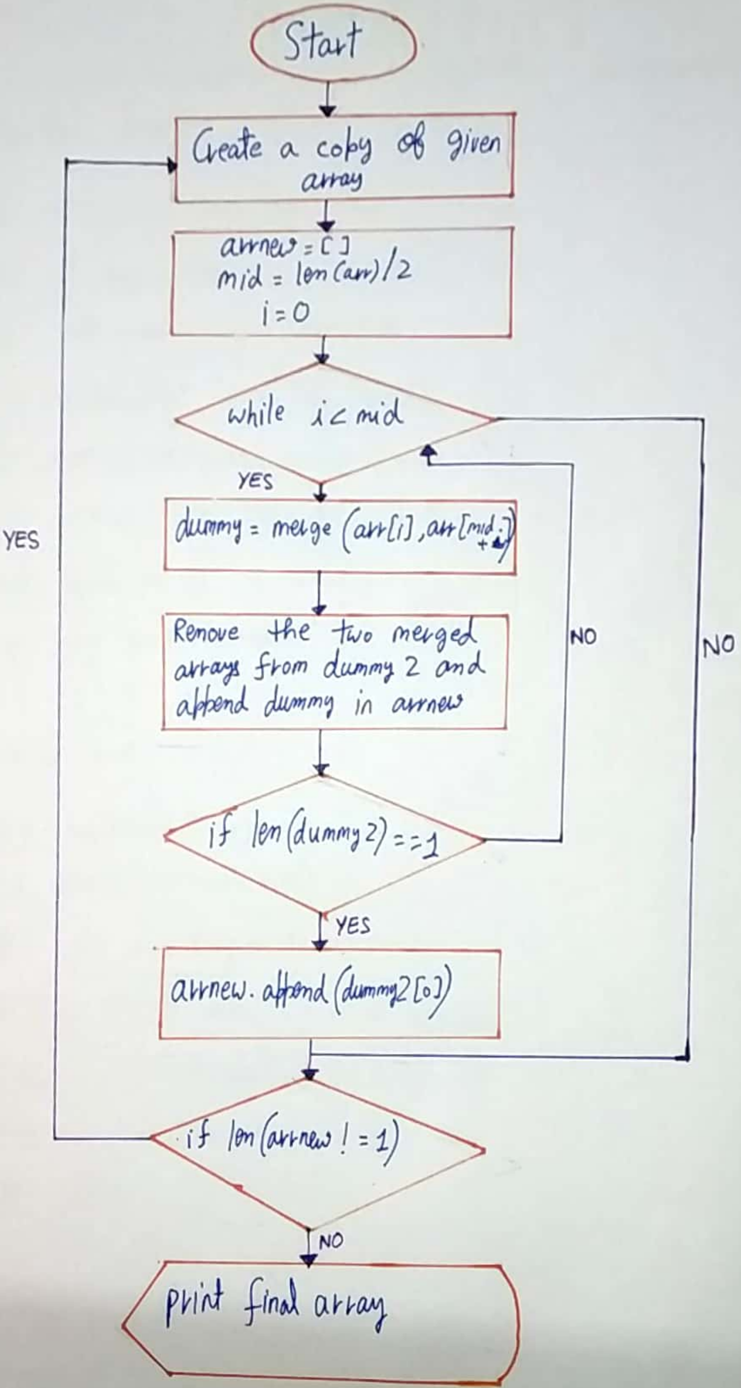


FLOWCHART - DISPLAYING HEAP SORT



Scanned with CamScanner

◉ FLOWCHART - DISPLAYING M-METHOD



EXPLANATION OF ALGORITHM

⊙ MERGE SORT:

Step 1: MERGE_SORT(arr, beg, end)

Step 2: if beg < end

Step 3: set mid = (beg + end)/2

Step 4: MERGE_SORT(arr, beg, mid)

Step 5: MERGE_SORT(arr, mid+1, end)

Step 6: MERGE (arr, beg, mid, end)

Step 7: end of if

Step 8: END MERGE_SORT

⊙ HEAP SORT:

Step 1: HeapSort (arr)

Step 2: BuildMaxHeap (arr)

Step 3: for i = length (arr) to 2

Step 4: swap arr[1] with arr[i]

Step 5: heap-size[arr] = heap-size[arr] ? 1

Step 6: MaxHeapify (arr, 1)

Step 7: End

⊙ BuildMaxHeap(arr) :

Step1: BuildMaxHeap(arr)
Step2: heap-size(arr) = length(arr)
Step3: for i = length(arr)/2 to 1
Step4: MaxHeapify(arr, i)

⊙ MaxHeapify (arr, i) :

Step1: MaxHeapify(arr, i)
Step2: L= left (i)
Step3: R= right (i)
Step4: if L ? heap-size [arr] and arr[L] > arr [i]
largest = L
Step5: else
largest = i
Step6: if R ? heap-size [arr] and arr [R] > arr [largest]
largest = R
Step7: if largest != i
Step8: swap arr[i] with arr [largest]
Step9: MaxHeapify (arr, largest)
Step10: End

⊙ MEAP SORT :

Step 1: We will take the input of 'n' elements in the list.

Step 2: Then divide the main list into multiple smaller lists of 3 elements each.

Step 3: Select a small list.

Step 4: Apply HeapSort Algorithm in the selected small list.

Step 5: if small lists are sorted
  select pairs of list
 else
  Go to step 3

Step 6: After selecting the pair of lists in step5 apply M-method.

Step 7: If 1 list remains then we would print final list,
 otherwise goto step 6

Step 8: Terminate

MAIN PROGRAM


```
# FINAL ALGORITHM

def heapify (arr, n, i):
    largest = i  # initialize largest as root
    l = 2 * i + 1  # left = 2*i + 1
    r = 2 * i + (1+1)  # right = 2*i + 2

    # See if left child of root exists and is
    # greater than root
    if l < n and arr[i] < arr[l]:
        largest = l
    # See if right child of root exists and is
    # greater than root
    if r < n and arr [largest] < arr [r]:
        largest = r

    # change root, if needed

    if largest != i:
        arr[i], arr[largest] = arr[largest], arr[i]  # swap

        # Heapify the root

        heapify (arr, n, largest)

# The main function to sort an array of given state

def heapsort (arr):
    n = len (arr)
    # Build a maxheap
```

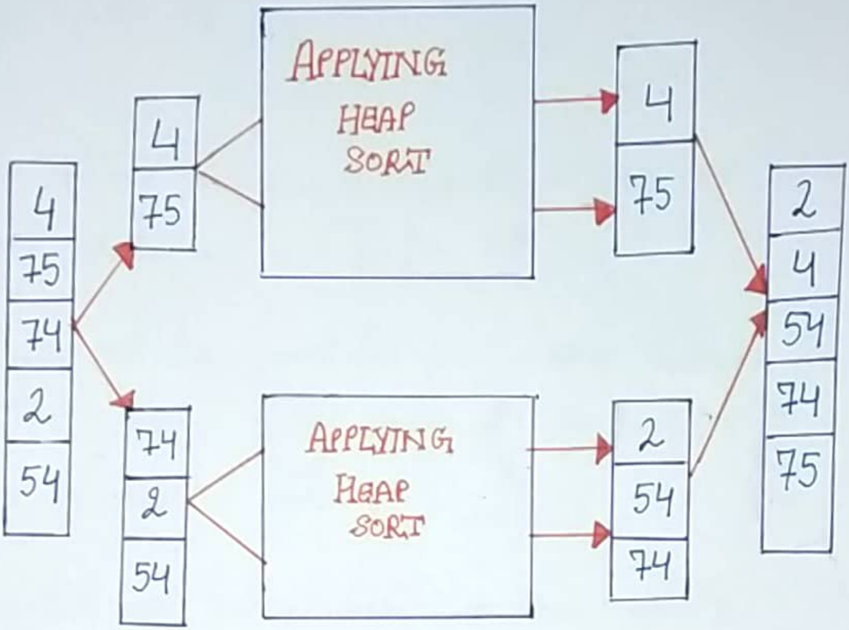
```
for i in range (n,-1,-1):
    heapify (arr, n, i)
# One by One extract elements
for i in range (n-1, 0, -1):
    arr[i], arr[0] = arr[0], arr[i]  # swap
    heapify (arr, i, 0)

rawlist = [ ]
print ("ENTER THE SIZE OF RAW LIST \n")
N=int(input ())

# taking raw data input
for i in range (0,N):
    rawlist.append (int (input ()))

LIST = [ ]
f = N % 3
if (f==0):
    x = N//3
    count = 0
    for i in range (0,x):
        k = 3*count
        dummy = [ ]
        for j in range (k, k+3):
            dummy.append (rawlist [j])
        count = count + 1
        LIST.append (dummy)
```

```
dummy = []
for j in range (k+3, k+3+f):
    dummy.append (rawlist [j])
LIST.append (dummy)
print (LIST) #LIST IS READY TO BE SCORED WITH HEAP SORT
L2=[]
k=[]
for i in LIST:
    heapifySort (i)
    L2.append (i)
print (L2)
for i in range (1, len (L2)):
    L2[0].extend (L2[i])
    L2[0].sort ()
print (L2[0])
```

OUTPUT

```
ENTER THE SIZE OF RAW LIST
7
45
23
67
98
1
0
33
[[45, 23, 67], [98, 1, 0], [33]]
[[23, 45, 67], [0, 1, 98], [33]]
[0, 1, 23, 33, 45, 67, 98]
```

DRY RUN



Working Logic of the Proposed Algorithm

ANALYSIS OF TIME COMPLEXITY

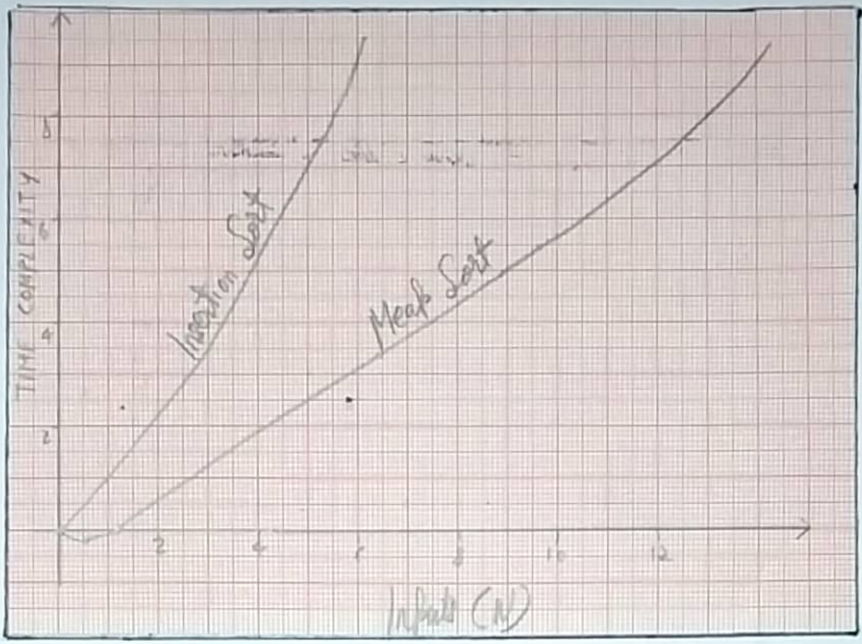
→ We compare our hybrid algorithm diligently with bubble sort, insertion sort and selection sort.

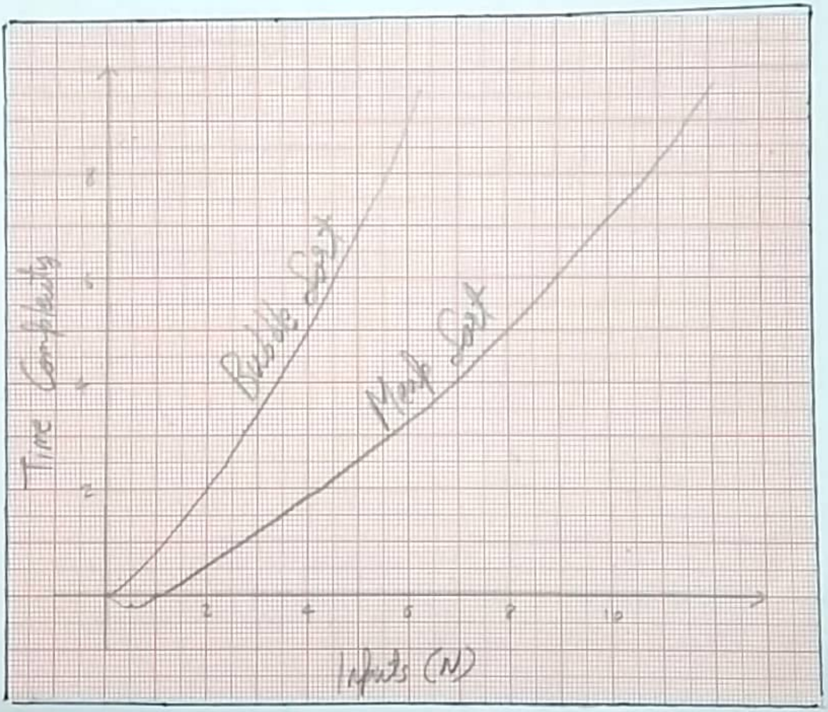
→ Bubble sort has a worst case and average complexity of $O(n^2)$, where n is the number of items being sorted or the number of inputs

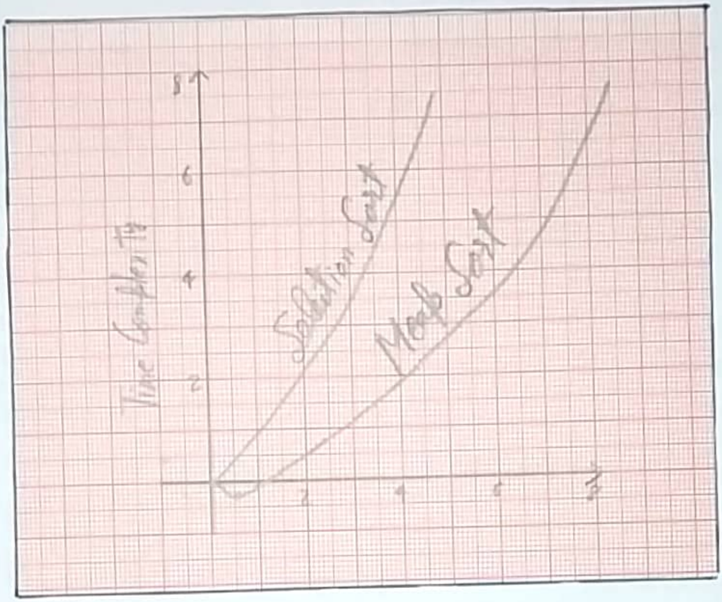
→ Selection sort has a best-case time complexity of big omega (n^2) as well as its worst case time complexity is also $O(n^2)$

→ Insertion sort has a best-case time complexity of big omega of n and worst-case time complexity of $O(n^2)$

→ Graphs of following sorting techniques have been depicted and from that we can conclude that meap sort has time complexity of $O(n\log n)$.


TIME COMPLEXITY
Insertion Sort
Heap Sort
Inputs (N)


Time Complexity
Bubble Sort
Heap Sort
Inputs (N)



Scanned with CamScanner

APPLICATIONS

As the algorithm has already proved itself well in the case of integer values, this could be applied to other complex data types and its performance could be evaluated. This could be helpful in sorting the character strings and be applied in, for example, contacts in mobile phones, words in dictionary etc.

The two dimensional matrices are being used to store and represent massive data in many fields such as engineering design and management. Efficient two-dimensional sorting algorithms are needed when these data are sorted by computers. The proposed algorithm could act as a base for the development of effective two dimensional sorting algorithms that could serve the purpose.

RESULT

The Project proves that the proposed hybrid merge-heap algorithm [meap Sort] works remarkably well in worst-case scenarios.

The flowchart, explained algorithm and time-complexity analysis proves that although the complexity of meap sort is same as the complexity of merge and heap that is $O(n \log n)$, but the performance is exceptionally well.

Further, the dry run alongwith the applications of the proposed meap sort technique is also mentioned.

References:

1. Katajainen, Jyrki; Pasanen, Tomi; Teuhola, Jukka (1996). "Practical in-place mergesort". Nordic Journal of Computing. 3. pp. 27–40. ISSN 1236-6064. Archived from the original on 2011-08-07. Retrieved 2009-04-04. Also Practical In-Place Mergesort.
2. Knuth, Donald (1998). "Section 5.2.4: Sorting by Merging". Sorting and Searching. The Art of Computer Programming. 3 (2nd ed.). Addison-Wesley. pp. 158–168. ISBN 0-201-89685-0.
3. Kronrod, M. A. (1969). "Optimal ordering algorithm without operational field". Soviet Mathematics - Doklady.
4. Data structures and algorithm analysis in C++ by Mark Allen Weiss (3 edition)
5. Data structures a programming with c by Dharmendra singh Kushwaha,
6. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. Introduction to Algorithms, Second Edition. MIT Press and McGraw-Hill, 2001. ISBN 0-262-03293-7. Problem 2-2, pg.40.
7. https://www.academia.edu/16724528/Proposal_of_a_Two_Way_Sorting_Algorithm_and_Performance_Comparison_with_Existing_Algorithms.
8. Peter Brass, Advanced Data Structures, Cambridge University Press, 2008, ISBN 978-0521880374
9. Donald Knuth, The Art of Computer Programming, vol. 1. Addison-Wesley, 3rd edition, 1997, ISBN 978-0201896831
10. Dinesh Mehta and Sartaj Sahni, Handbook of Data Structures and Applications, Chapman and Hall/CRC Press, 2004, ISBN 1584884355
11. Niklaus Wirth, Algorithms and Data Structures, Prentice Hall, 1985, ISBN 978-0130220059
12. Alfred Aho, John Hopcroft, and Jeffrey Ullman, Data Structures and Algorithms, Addison-Wesley, 1983, ISBN 0-201-00023-7
13. G. H. Gonnet and R. Baeza-Yates, Handbook of Algorithms and Data Structures - in Pascal and C, second edition, Addison-Wesley, 1991, ISBN 0-201-41607-7
14. Ellis Horowitz and Sartaj Sahni, Fundamentals of Data Structures in Pascal, Computer Science Press, 1984, ISBN 0-914894-94-3
15. Animated Sorting Algorithms: Heap Sort at the Wayback Machine (archived 6 March 2015) – graphical demonstration
16. Courseware on Heapsort from Univ. Oldenburg - With text, animations and interactive exercises
17. NIST's Dictionary of Algorithms and Data Structures: Heapsort
18. Heaps and Heapsort Tutorial by David Carlson, St. Vincent College
19. Sorting revisited by Paul Hsieh
20. A PowerPoint presentation demonstrating how Heap sort works that is for educators.