International Journal of Hybrid Information Technology
Vol. 10, No. 7 (2017), pp.23-30
http//dx.doi.org/10.14257/ijhit.2017.10.7.03

Binomial Heap Sorting – A New Sorting Algorithm

Ashish Tayal[1], Kuldeep Tanwar[2], Gaurav Dubey[3]
[1,2,3]Department of Computer Science & Engineering,
AMITY School of Engineering & Technology, Noida, India
[1]ashish.tayal@yahoo.com, [2]tanwarkd@gmail.com, [3]gdubey@amity.edu

Abstract

In computer science, one of the fundamental issue is to arrange the elements in some logical order. Various algorithms have been developed over the years to handle this process of ordering the elements (sorting) having their own running time, efficiency and simplicity. Sorting problem has attracted a great deal of research because efficient sorting is important to optimize the use of other algorithms. This report presents a new sorting algorithm, using the data structure Binomial Heap, called Binomial Heap sort algorithm which will give an innovative sorting scheme through which we can arrange the given elements in some specific order either in increasing order or in decreasing order. Binomial Heap sort results in minimum complexity over Binary Heap sort as far as frequency of inbuilt operations is concerned. In this report, we consider all the cases whether the input elements given in an array are in descending order, ascending order or randomly distributed. Through this paper, we are proposing a new sorting algorithm called Binomial Heap sort. Its time complexity is $O(n \lg n)$.

***Keywords**: Sorting, Binomial Heap, Union, Extract Max, Binomial Heap Sort*

1. Introduction

Sorting is a technique that puts the given sequence of elements in some mathematical or logical order or we can say that sorting algorithms are those which put all the elements of the list in a specific order. Sorting algorithms may be comparison based sort (which requires comparisons) or non-comparison based sort (which do not requires comparisons) but the more popular is comparison based sort. Very fast computers exist now a days but there is also a limitation on processing because they are not able to solve all the problems. Memory is also not very costly but not free of cost. Today we have computers which have a very large RAM. But the running time will always remain a bounding resource. Various sorting algorithms exists with running time ranges from $O(n)\,to\,O(n^2)$.

2. Definitions & Notations

2.1. Heap

A binary heap can be viewed as an array and we can represent them as almost complete binary tree. We will insert elements in a heap from left to right until the level is filled and similarly elements in a heap can be removed from right to left until the level is empty. Two kinds of heaps which exists are: Maximum-heap and Minimum-heap. In Maximum-heap, the information part of the node is greater than or equal to the information part of its children's (if exists). In Minimum-heap, the information part of the node is greater than or equal to the information part of its children's (if exists). The general approach to perform Heap sort is

1. As array is given as input, first we create a Max-heap.
2. Exchange the root element with the last level element.

ISSN: 1738-9968 IJHIT

3. Remove the last node.
4. Maintain the Max-heap property to again convert it into the Max-heap.
5. Repeat the above procedure until there are no more elements.

[2]*MAX-HEAPIFY(A, i)*

1. l ← LEFT(i)
2. r ← RIGHT(i)
3. if l ≤ heap-size[A] and A[l] > A[i]
4. then largest ← l
5. else largest ← i
6. if r ≤ heap-size[A] and A[r] > A[largest]
7. then largest ← r
8. if largest ≠ i
9. then exchange A[i] ↔ A[largest]
10. MAX-HEAPIFY(A, largest)

2.2. Binomial Heap

A Binomial heap Consists of several Binomial trees satisfies the minimum-heap property i.e. the key of a node is greater than or equal to the key of its parent or we can say that all the Binomial trees are min-heap ordered. Binomial heap can be represented using linked list. For any non-negative integer k, there is at most one binomial tree in H whose root has degree k. We can say that a Binomial heap consists of unique binomial trees. E.g. if n = 13 (let us suppose that there are 13 nodes in Binomial tree), then just write the binary representation of 13 which is 1101, thus a Binomial heap consists of three binomial trees, B_0, B_2, and B_3 respectively.

2.3. Binomial Tree

The Binomial tree can be defined as follows

1. Binomial tree is denoted as B_k.
2. Binomial tree is an ordered tree.
3. Binomial tree can be defined recursively.
4. Binomial tree B_0 consists of a single node.
5. The binomial tree B_k contains two copies of B_{k-1} trees that are attached together in such a way that the root node of one tree is the leftmost child of the root node of the other.
6. Every binomial tree in Binomial heap H satisfies the minimum-heap property: the information part of a node is greater than or equal to the information part of its parent. If y represents a node in heap, then
    - p[y] refers to the parent of node y
    - child[y] refers to the leftmost child of node y
    - sibling[y] refers to the sibling of node y
    - key[y] denotes the key value of node y
    - degree[y] denotes the number of children of node y

3. Operations on Binomial Heaps

3.1. Creating a new Binomial Heap (CREATE-BINOMIAL-HEAP ())

The procedure CREATE-BINOMIAL-HEAP() creates an empty binomial heap.

head[H]=NIL

3.2. Uniting two Binomial Heaps (BINOMIAL-HEAP-UNION (H_1, H_2))

This operation unites the two heaps H_1 and H_2 such that the resulting binomial heap consists of all the elements of H_1 and H_2 and destroy both H_1 and H_2. The running time of BINOMIAL-HEAP-UNION is $T(n) = O(\lg n)$.

This procedure starts by creating an empty binomial heap. Then, **BINOMIAL-HEAP-MERGE(H_1, H_2)** procedure merges the given two Binomial heaps according to the degree of the nodes present in the root list of both the heaps and the address of first node is stored in head[H]. Three pointer variables prev-x, x and next-x are used which contains the address of three successive nodes.

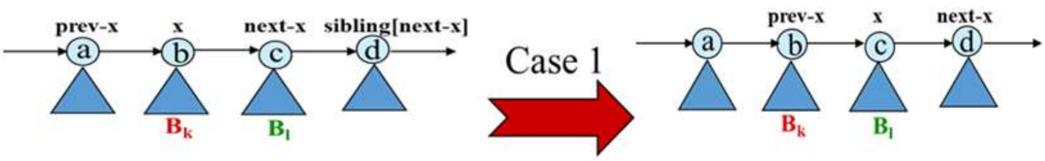
[2]BINOMIAL-HEAP-UNION (H_1, H_2)

```
H = Create a new Binomial Heap
head[H] = Binomial Heap Merge according to degree of nodes
Remove both the heaps H1 and H2
if head[H] == NULL
   then return NULL HEAP
prev-y = NIL
y = head[H]
next-y = sibling[y]
while next-y ≠ NIL
        do if (degree[y] ≠ degree[next-y]) or  (sibling[next-y] ≠ NIL and
                degree[sibling[next-y]] = degree[y])
                then prev-y = y
                        y = next-y
                else if key[y] ≤ key[next-y]
                        then sibling[y] = sibling[next-y]
                                BINOMIAL-LINK(next-y, y)
                        else if prev-y == NIL
                                then head[H] = next-y
                                else sibling[prev-y] = next-y
                              BINOMIAL-LINK(y, next-y)
                              y = next-y
              next-y = sibling[y]
return H
```

There are four cases for uniting two binomial heaps.

Case 1

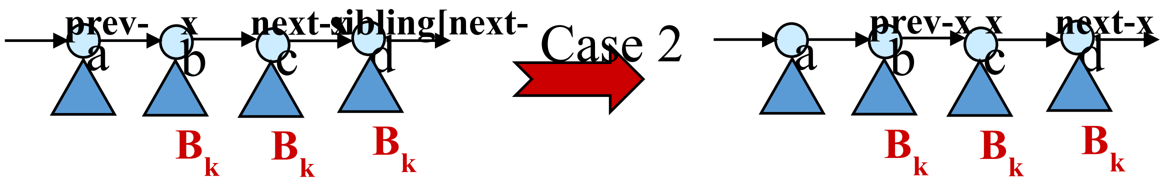
degree[y] ≠ degree [next-y]) i.e. when y is the root node of a binomial tree B_k and next-y is the root node of binomial tree B_l for some l > k. Then we move the pointers y and next-y one position further down the root list.



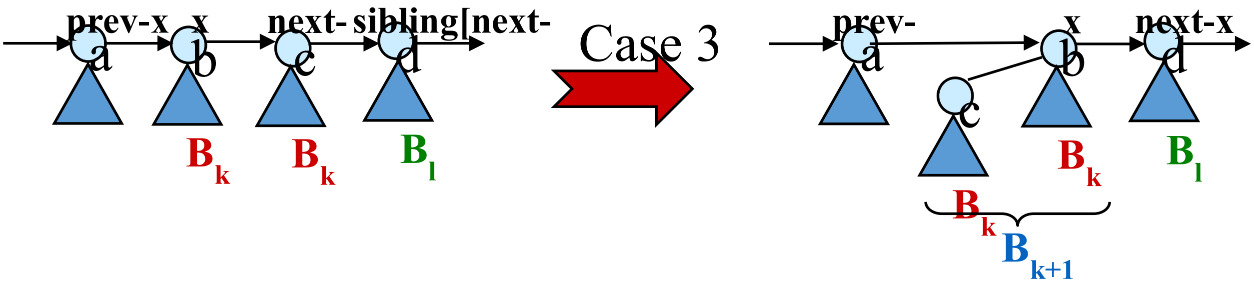
Case 2

degree[y] = degree[next-y] = degree[sibling[next-y]]

We handle this case in the same manner as case 1: again we just move the pointers one position further down the root list.

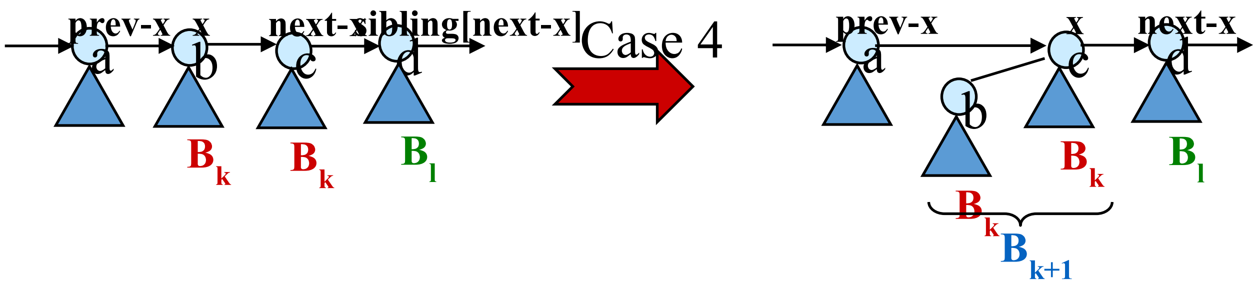


Case 3
degree[y] = degree[next-y] ≠ degree[sibling[next-y]] and key[y] ≤ key[next-y]



We remove next-y from the root list and link it to y, creating a B_{k+1} tree.

Case 4
degree[y] = degree[next-y] ≠ degree[sibling[next-y]] and key[y] > key[next-y]



We remove next-y from the root list and link it to next-y, again creating a B_{k+1} tree**.**

3.3. Linking two Binomial Heaps (BINOMIAL-LINK (i, j))

The Binomial link procedure links two binomial trees whose roots have same degree.

[2]*BINOMIAL-LINK (i, j)*

1. *p[i]=j*
2. *sibling[i]=child[j]*
3. *child[j]=i*
4. *degree[j]=degree[j]+1*

3.4. Inserting a Node (BINOMIAL-HEAP-INSERT (H, y))

This procedure inserts a node y into the Binomial heap H.

3.5. Extracting/Removing the Node Containing Minimum Information BINOMIAL-HEAP-EXTRACT-MIN (H)

This operation extracts/removes the node containing the minimum information part from Binomial heap H and returns a pointer or gives the address of the extracted node. The running time of BINOMIAL-HEAP-EXTRACT-MIN (H) is $T(n) = O(\lg n)$.

BINOMIAL-HEAP-EXTRACT-MIN (H)

1. Traverse the root list of Binomial heap and find the node with the minimum information part, designated as y and eliminate it from the root list of H. During traversal we are just comparing the key value of nodes present in the root list.
2. H'= CREATE-BINOMIAL-HEAP(). Using this procedure, an empty heap is created whose head points to NULL.
3. All y's children are to read in reverse direction (putting all the children of y in new root list in increasing order or Binomial trees) and set head[H'] to point to the head of the resulting list.
4. H=BINOMIAL-HEAP-UNION (H_1,H_2)
5. return y

4. Proposed Algorithm of BINOMIAL HEAP SORT

In Binomial heap sort, we will assume that array of elements is given as input. The array will be sorted using the Binomial heap data structure. The Binomial Heap Sort procedure works as follows:

1. Iterate the array for every element and create a node corresponding to each element i.e. each element of an array will be represented by a node of the Binomial tree.
2. Now use the procedures Create Binomial Heap, Binomial Heap insert and Binomial Heap as written above and a binomial heap (min-heap ordered) is created.
3. Now use BINOMIAL-HEAP-EXTRACT-MIN(H) which returns an address of the node having the minimum information part say y. Now put the minimum key value (key[y]) into the array starts from the lower index (say index=1) and increment the index. Again repeat the same procedure and put the corresponding key value into the array at the next index (say index=2) and continue in the same way until the Binomial heap becomes empty. Finally the output array is sorted.

BINOMIAL-HEAP-SORT (A)

```
1.  for i=1 to length[A]
2.      do create a node y corresponding to the element A[i]
3.          H = Create Binomial Heap
4.          p[y]=NIL
5.          child[y]=NIL
6.          sibling[y]=NIL
7.          degree[y]=0
8.          head[H']=y
9.          H= Binomial Heap Union of H and H'
10.  index=1
11. while(head[H] ≠ NIL)
12.        do x= BINOMIAL-HEAP-EXTRACT-MIN (H)
13.            if(x ≠ NIL)
14.              then A[index]=key[x]
15.            index=index+1
16. for a=1 to length[A]
17.     do print A[a]
```

5. Analysis of Binomial Heap Sort

Initially create a Binomial Heap from the input array inside for loop in steps 1-9 which takes a total time of $O(n \lg n)$. Steps 11-15 constitute a while loop which executes ***n*** times since the total elements are '***n***'. Inside the while loop BINOMIAL-HEAP-EXTRACT-

MIN(H) procedure executes ***lgn*** times so the total time for step 11-15 is $O(n\lg n)$. Thus, the total running time of BINOMIAL-HEAP-SORT is $O(n\lg n)$.

6. Comparison between Heapsort and Binomial Heap

The below table shows the frequencies of the procedures Binomial Heap Extract Min, Binomial Heap Union and Binomial Heap Merge.

Table 1. Comparison of Frequencies

S.No	No. of Elements	EXTRACT MIN FREQUENCY	UNION FREQUENCY	MERGE FREQUENCY
1	5	2	3	7
2	10	7	16	25
3	50	86	196	245
4	100	219	534	633
5	200	535	1398	1597
6	300	884	2306	2605
7	400	1267	3317	3716
8	500	1722	4448	4947
9	800	2931	7823	8622
10	1000	3938	10286	11285
11	2000	8870	23515	25514
12	5000	24809	68277	73276
13	8000	43462	117155	125154
14	10000	54613	150817	160816
15	15000	87259	238096	253095
16	20000	119221	328815	348814
17	25000	153762	421521	446520
18	30000	189511	515356	545355
19	35000	225106	611722	646721
20	40000	258437	708266	748265
21	45000	294349	805187	850186
22	50000	332518	903921	953920

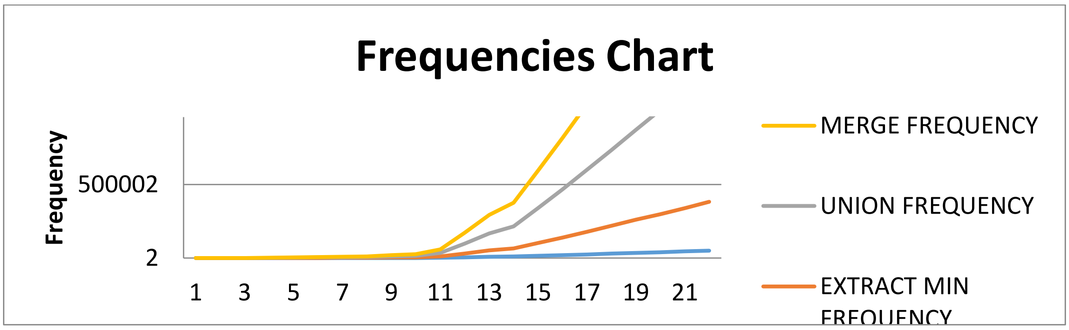


Figure 1. Frequencies Chart

The time taken by Heapsort depends on the height of the tree i.e. depends on how many times MAX-HEAPIFY(A, i) procedure executes since this procedure maintains the max-heap property while in Binomial Heapsort, the running time depends on how many times the BINOMIAL-HEAP-EXTRACT-MIN(H) procedure executes. The below Table 2 shows the comparison between frequencies of MAX-HEAPIFY(A, i) and BINOMIAL-HEAP-EXTRACT-MIN(H) procedure.

Table 2. MAX-HEAPIFY Frequency vs. BINOMIAL-HEAP-EXTRACT-MIN Frequency

S.No.	No. of Elements	Heapsort (MaxHeapify Frequency)	Binomial Heap Sort (Extract Min Frequency)
1	5	13	2
2	10	33	7
3	50	270	86
4	100	627	219
5	150	1033	369
6	200	1452	535
7	250	1916	739
8	300	2356	884
9	350	2836	1067
10	400	3314	1267
11	450	3787	1481
12	500	4274	1722
13	600	5302	2064
14	700	6367	2478
15	800	7433	2931
16	900	8454	3408
17	1000	9570	3938
18	2000	21142	8870
19	3000	33574	13835
20	4000	46342	19734
21	5000	59671	24809
22	6000	73153	30663
23	7000	86850	36635
24	8000	100703	43462
25	9000	114852	48825
26	10000	129226	54613
27	15000	202429	87259
28	20000	278223	119221
29	25000	356069	153762
30	30000	434814	189511
31	35000	514763	225106
32	40000	596612	258437
33	45000	679196	294349
34	50000	762175	332518

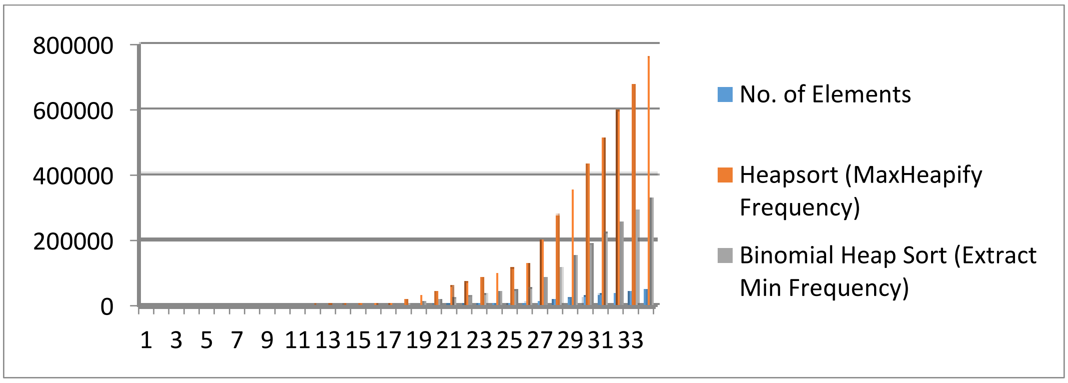


Figure 2. Frequencies Chart

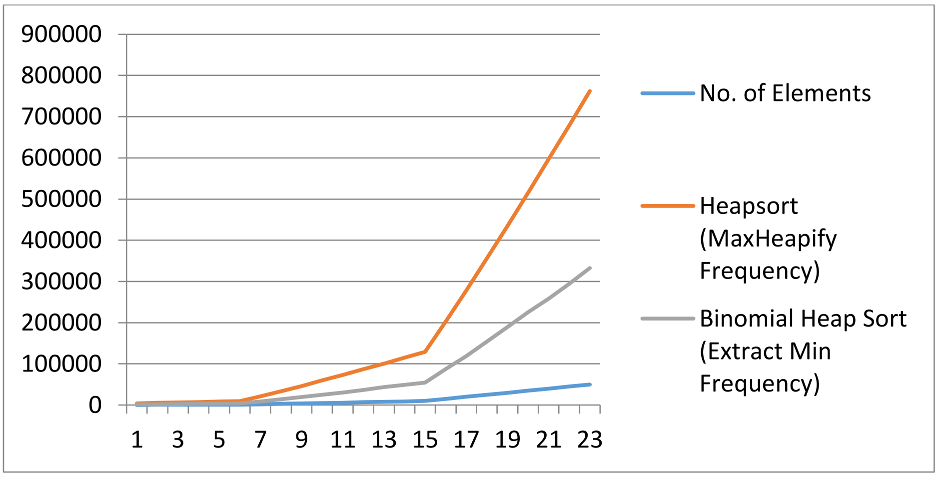


Figure 3. Frequencies Chart

7. Conclusion

Binomial Heap Sort is based on merging of Binomial trees. The height of resultant Binomial tree is significantly less than the height of the tree created in Binary Heap sort. The time complexity of this new proposed algorithm is of the order which is equivalent to the other existing sorting algorithms. Binomial Heap has a special characteristic that merging of heaps is of the order which is a major advantage over traditional heaps.

This new proposed sorting algorithm is likely to benefit wherever Binomial heaps are used.

References

[1] A.V. Aho, J.E. Hopcroft and J. D. Ullman, "Data Structures and Algorithms", Addison-Wesely, **(1983)**.

[2] CORMEN, H. Thomas, E.L. Chrles and L.R. Ronald, "Introduction to algorithms", MIT Press and Mc. Graw- Hill, **(2003)**.

[3] D.L.Shell "A High-Speed Sorting Procedure", Communications of the ACM, vol. 2, **(1959)**, pp. 30-32.

[4] J. Munro and V. Raman, "Sorting with minimum data movement.J.Algorithms, vol. 13, **(1992)**, pp. 374-393.

[5] E. Horowitz, S. Sahani, and S. Rajsekran, Computer Algorithms. Computer Science Press, **(1998)**.

[6] D.E. Knuth, "The art of computer programming, Sorting and Searching", Addison-esley vol. 3, **(1999)**.

[7] R.Sedgwick.(1977). " The Analysis of Quicksort with Programs", Acta Informatica, 240- 267.

[8] R. Fadel", K.V. Jakobsen", J. Katajainen", J. Teuholab,1999, " Heaps and heapsort on secondary storage " Theoretical Computer Science, vol. 220, **(1999)**, pp. 345-362.

[9] C. D. Scott and R. E. Smalley, "Diagnostic Ultrasound: Principles and Instruments", Journal of Nanosci. Nanotechnology, vol. 3, no. 2, **(2003)**, pp. 75-80.

