RINGSUM Operation on Two sets of positive integers

Akash Anand, Parth Kataria, Shruti Nanda
iit2019015@iiita.ac.in, iit20190016@iiita.ac.in, iit2019017@iiita.ac.in
IV semester,Department of Information Technology
Indian Institute of Information Technology, Allahabad

Abstract-This paper contains the algorithms to perform only node operation of ringsum operation on two sets of positive integers which represents nodes of the two graphs.We will discuss time complexities of all algorithms and differences in them.

1. INTRODUCTION

Definition - Given two graphs G1 = (V1, E1) and G2 = (V2, E2) we define the ring sum G1 ⊕ G2 = (V1 ∪ V2,(E1 ∪ E2) − (E1 ∩ E2)) with isolated points dropped. So an edge is in G1 ⊕ G2 if and only if it is an edge of G1 , or an edge of G2, but not both.

Node Operation-(V1 U V2)
Edge Operation-(E1 ∪ E2) − (E1 ∩ E2)
This report further contains -

1. Algorithm Design and Analysis
2. Experimental Study and Complexity
3. Conclusion

2. ALGORITHM DESIGN

We designed three algorithms to perform union of two sets of positive integers.Basic steps for designing the algorithm are-

1. Create two arrays each representing a set of elements with sizes n and m where (n,m>=1000).
2. Using rand() function to generate positive integers which are elements of the sets.
3. Now pass the two arrays along with their sizes as arguments in the functions of following algorithms to print the union of the two arrays.

Algorithm 1-

```
void brute(int n, int m, int set1[], int set2[])
{
        int i,j;
        for(i=0;i<n;i++)
                printf("%d ",set1[i]);
        for(i=0;i<m;i++)
        {
                for(j=0;j<n;j++)
                        if(set2[i]==set1[j])
                                break;
                if(j==n)
                        printf("%d ",set2[i]);
        }
        printf("\n");
}
```

Analysis-We print all the elements of the larger set and all elements of the smaller set which are not present in the larger set using brute-force searching.

Algorithm 2-

```
int cmp(int a,int b)
{
        if(a<b)
                return a;
        else
                return b;
}

void union(int n,int m,int set1[],int set2[])
{
        qsort(set1,n,sizeof(int),cmp);
        qsort(set2,m,sizeof(int),cmp);
```

```
        int i = 0, j = 0;
    while (i < m && j < n) {
      if (set1[i] < set2[j]) {
        printf("%d ", set1[i]);
        i++;
      }
      else if (set2[j] < set1[i]) {
                printf("%d ", set2[j]);
                j++;
        }
      else {
        printf("%d ", set2[j]);
        i++;
        j++;
      }
    }
    while (i < m)
    {
      printf("%d ", set1[i]);
      i++;
    }
    while (j < n)
          {
      printf("%d ", set2[j]);
      j++;
    }
}
```

Analysis-In this algorithm we first sort the two arrays and using two pointers we compare elements of both the sets.

ALgorithm 3-

Sorting and searching

```
Void
union(int n, int m, int set1[], int set2[])
If m<n
  Sort(set2[]);
Printf(set2[]);
For i=0 to n
If Binary search(set[i])==false
     Print(set[i]);
Else
  Sort(set1[]);
Printf(set1[]);
For i=0 to n
  If Binary search(set[i])==false
     Print(set2[i]);
```

Analysis: we first find out which array is smaller then we will sort that array and print it then we will apply binary on each element of another array and if the element is not present in the previous array then we will print it.

3. EXPERIMENTAL STUDY AND ANALYSIS

A.brute()

In the first loop the complexity is directly to the number of elements in the set(n).In the second nested loop we look for common elements so the worst case depends on n*m.Therefore,worst case time complexity will be O(n+n*m).

t_{worst} : O(n+n*m).

B.union()

In this algorithm we first sort both the arrays therefore in the worst case it will take nlogn + mlogm and the remaining two pointer algorithm will take n+m operations.Therefore,worst case time complexity will be O(n(logn) + m(logm) +n +m).

t_{worst} : O(n(1+logn)+m(1+logm)).

C. Sorting And Searching

In this Algorithm We first find out which array is smaller and sort it in O(Nlogn) complexity. Then we will apply binary search on each element of another array in O(mlogn) and if the element is present in the first array then we will ignore it and print the element which is not present in the first array.

S no.	N	M	Algorithm1	Algorithm2	Algorithm 3
1	5	5	30	33.219280	38.219280
2	20	20	420	212.87712	232.87712
3	40	40	1640	505.75424	545.75424
4	50	50	2550	664.38561	714.38561

5	70	70	4970	998.09962	1068.0996
6	100	100	10100	1528.7712	1628.7712
7	1000	1000	1001000	21931.568	22931.568
8	1000	10000	10001000	153842.90	121623.62

4. TIME COMPLEXITY

The algorithms were tested against positive random sets of variable sizes.The result thus obtained from this experiment is given below:

For brute():
Best case : Ω(n+m)
Average case : θ(n+n*m)
Worst case : O(n+n*m)

For union():
Best case : Ω(n+m)
Average case : θ(n(logn + 1) +m(logm + 1))
Worst case : O(n(logn + 1) +m(logm + 1)

For Searching and sorting
Best case : Ω(mlogm+n)
Average case : θ((m+n)log(min(m,n)))
Worst case : O((m+n)log(min(m,n)))

Figure1:Time Complexity of Algorithm1

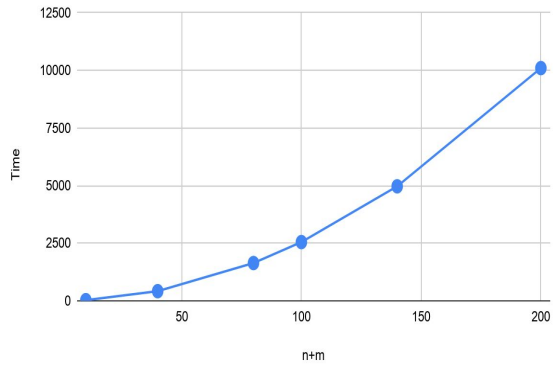


Figure2:Time Complexity of Algorithm2

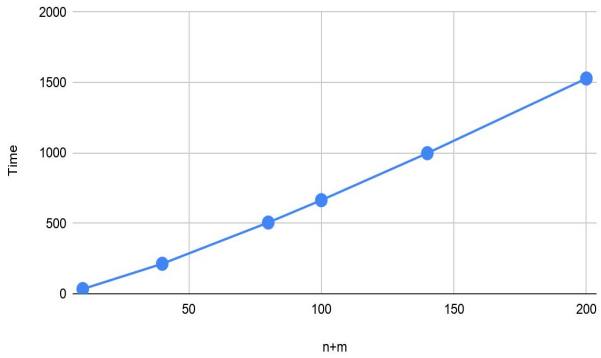
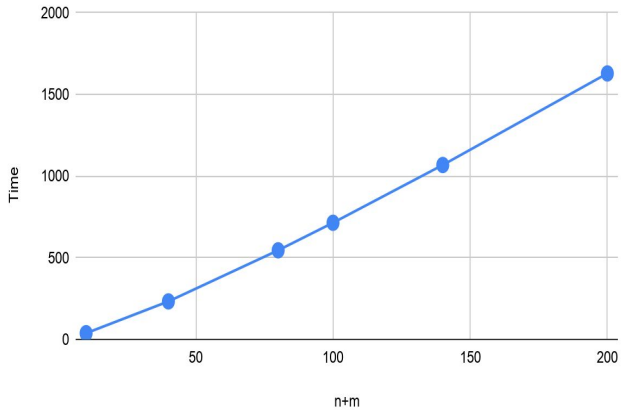
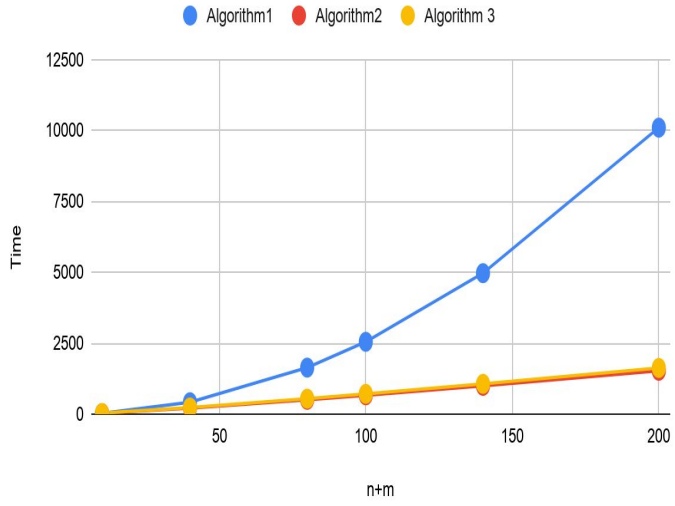


Figure 3:Time Complexity of Algorithm3



Comparison Graph



5.CONCLUSION

After doing the analysis of different algorithms we concluded that time complexity will be minimum in brute force and union algo when both of the arrays will be the same and for the worst case it will be minimum in Searching and Sorting Algorithm.

6. REFERENCES

- https://www.geeksforgeeks.org/union-and-intersection-of-two-sorted-arrays-2/
- https://www.google.com/search?q=Ringsum+operation+on+two+sets+of+positive+integers&oq=Ringsum+oper&aqs=chrome.1.69i57j69i59l2.7684j0j7&sourceid=chrome&ie=UTF-8