CS 2810
ADVANCED PROGRAMMING LAB

IMPLEMENTATION OF BINOMIAL HEAPS

TEAM:
Madhavi Y *CS09B025*
D Vamsi Krishna *CS09B006*
M Uday Chowhan *CS09B034*

Files included:
1. bheap.cpp
2. bheap.h
3. rand.cpp

•bheap.h contains declarations of the functions and classes used in the binomial heap class.
•bheap.cpp has the definitions and the main function.
•rand.cpp outputs random values into input files

Description:
•The bheap is implemented using lists(nodes) and Nodeuni class
•Nodeuni class stores sibling of current Node
•Each node has data , child , right sibling,left sibling, parent , degree

Functions in the bheap class
•return_root() : This function returns the root of the binomialheap
•insert : This function inserts an element into the binomial heap
•extract_min : This function extracts the minimum from the binomial heap
•dec_key : This function decreses the value of the key at the specified Node
•del : This function deletes the first occurence of the given key from binomial heap

Functions outside the bheap class
•merge : Function which merges two binomial heaps
•link : Function which links two binomial heaps of same degree and forms a binomial heap of one degree greater than that.
•unioun : Function which makes a binomial heap from merged binomial heaps

Pseudocode for functions in the leftist_heap:

Pseudocode for merge operation
// *to merge two binomial heaps*

```
merge (P_h1,p_h2)
{
      push_vector .v1 <- nodes in head row of p_h1
      push_vector .v2 <- nodes in head row of p_h2
```

```
      while(it_v1<end.v1&&it_v2<end,v2)
      {
            if(*it_v1->deg < *it_v2->deg)
                  push_back.v(*it1)
            else
                  push_back.v(*it2)
      }
      if(it1_v1==v1.end)
            push all elements into v from vector v2
      else
            push all elements into v from vector v2

      point all adjacent nodes in vector
      return v.front
}
```

Psedocode for link operation

// *To link binomial heaps of same degree*

```
link (Node *y,Node *z)
{
      p[y]<-z
      sibling[y]<-child[z]
      child[z]<-y
      deg[z]++
}
```

Psedocode for extract_min operation

//*To extract an element from the binomial heap*

```
extarct_min(  )
{
      curr=head
      while(curr!=NIL)
      {
            if(min.key<curr.key)
                  min=curr
            curr=sib.curr
      }
      Nodeuni z(min)
      z_pre.sib=z_next
      min=uni(rev(min.child),head)
}
```

Pseudocode for union

// *To form a single binomial heap from given two binomial heaps*

```
union
(Node *h1 ,Node *h2)
```

```
      if h1==NIL
            return h2
      if h2==NIL
            return h1
      head=merge(h1,h2)
      intialzing z of Nodeuni class with head
      while(z.next!=NIL)
            if(deg(z_curr)!=deg(z_next) or
                  sib(z_next)!=NILand deg(sib(z_next))==deg(z_curr))
                  update z to next sibling
            else if key(z_curr)<key(z_next)
                  sib(z_curr)=sib(z_next)
                  link(z_next,z_curr)
                  else if z_pre=NIL
                        head=z_next
                        sib[z_pre]<-z_next
                        link(z,z_next)
                        z=z_next
            z_next=sib(z_curr)
      return head
}
```

psudeo code for

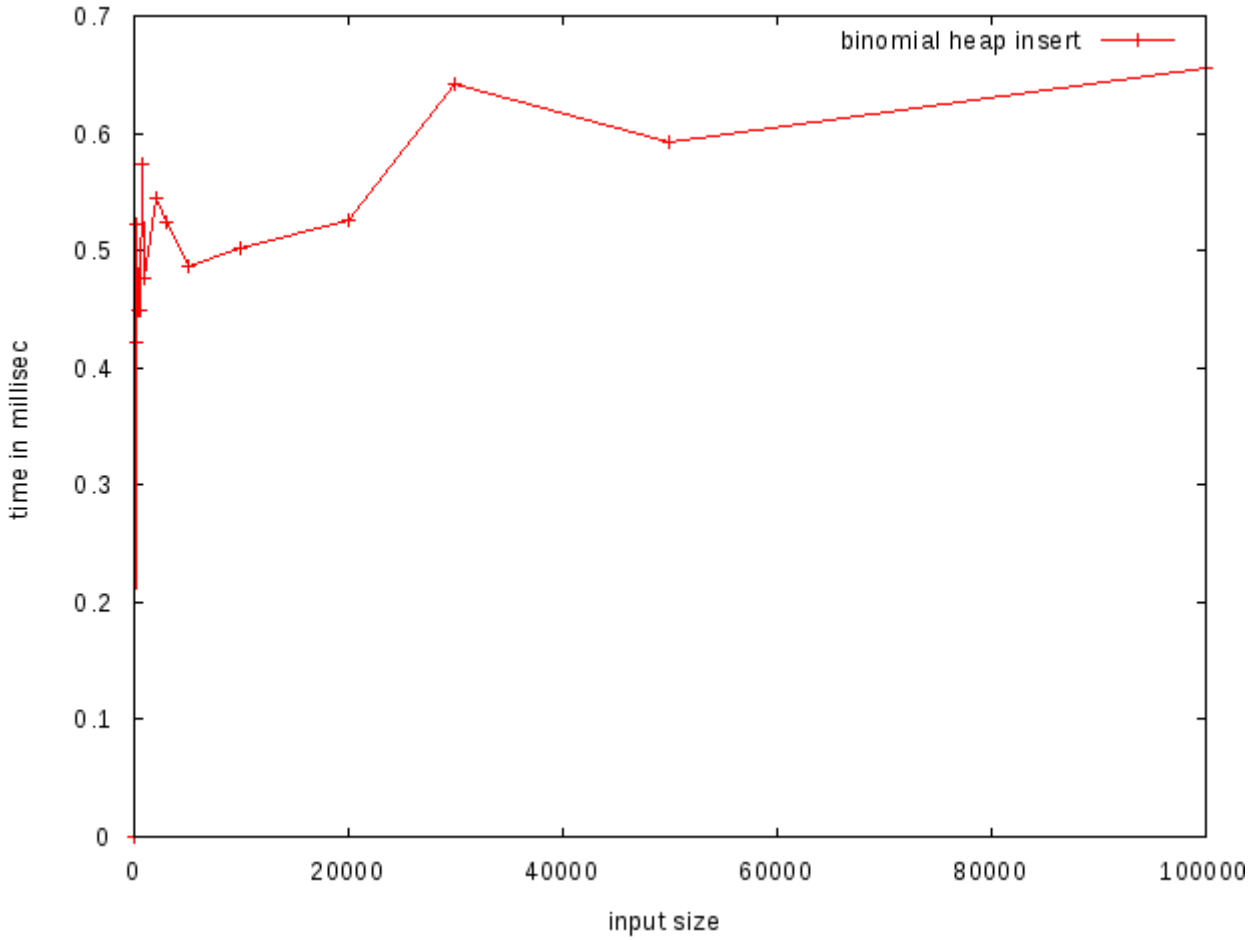
```
dec_key(Node *h ,num)
{
      h.key=num
      if(h.par.key<h.key&&h.par!=NIL)
            swap(h.par.key,h.key)
}

psudeocode for del(Node *h)
{
      dec_key(h,-9999999)
      extract_min()
}
```

.....

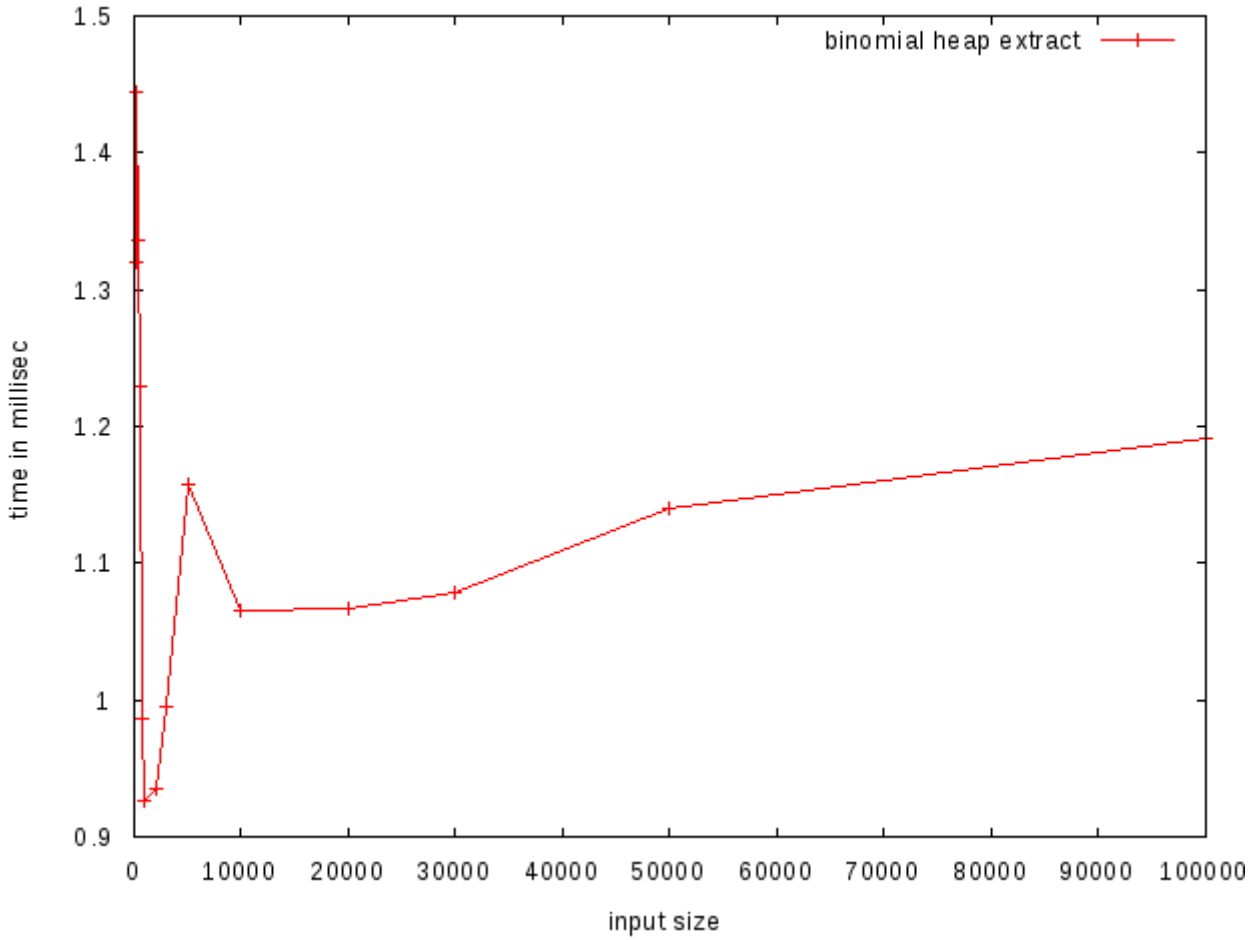
Time Vs size for insert operation :

size	time in ms
100	0.4211
200	0.5230
300	0.4491
500	0.4498
700	0.5731
1000	0.4771
2000	0.5443
3000	0.5243
5000	0.4873
10000	0.5015
20000	0.5255
30000	0.6421
50000	0.5917
100000	0.6549



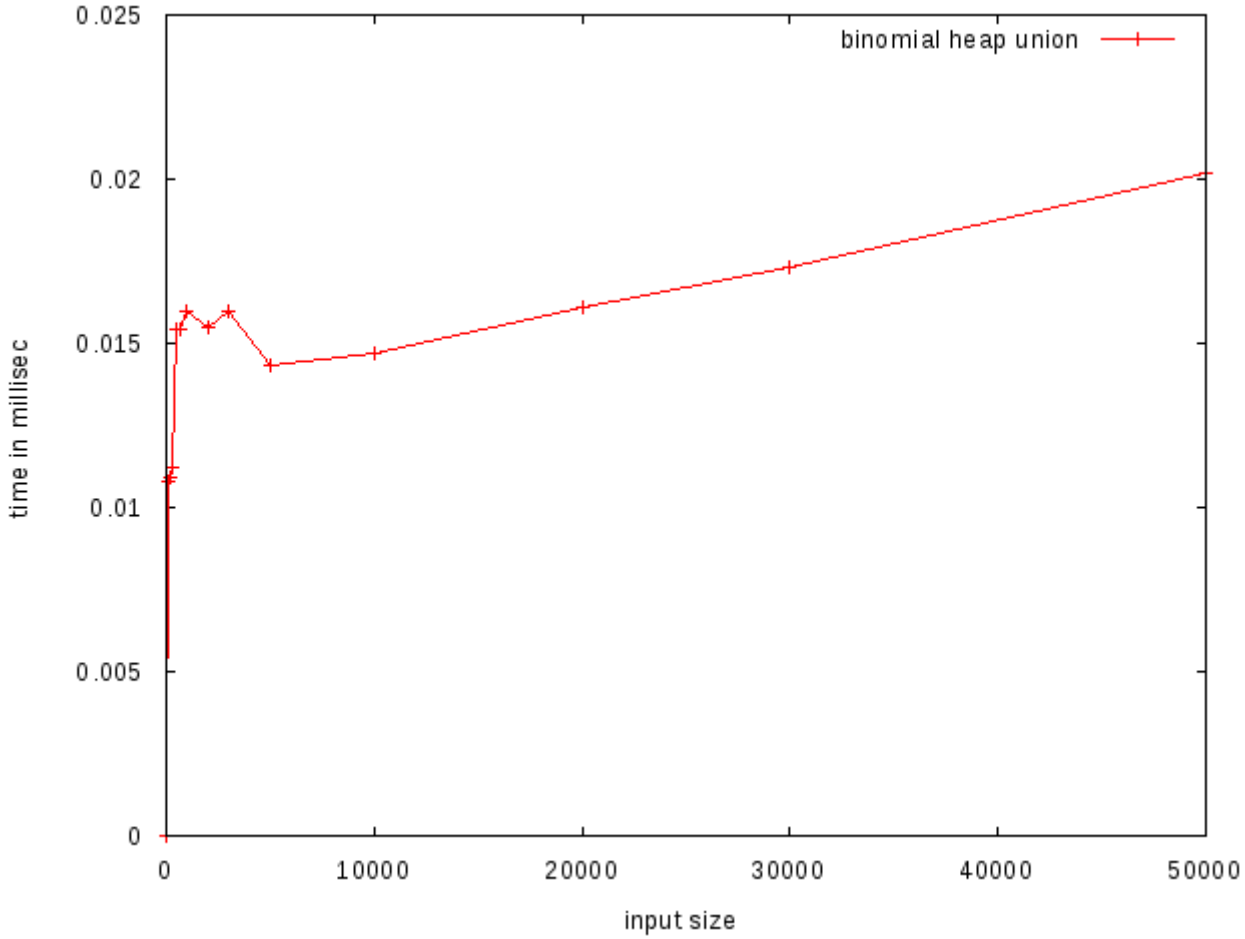
Time vs size graph for extrat operation :

size	time in ms
100	1.3203
200	1.4446
300	1.3355
500	1.2296
700	0.9861
1000	0.9269
2000	0.9346
3000	0.9958
5000	1.1575
10000	1.0658
20000	1.0667
30000	1.0786
50000	1.1397
100000	1.1913



Time vs size graph for union operation:

input size	time in ms
100	0.0108
200	0.0109
300	0.0112
500	0.0154
700	0.0154
1000	0.0160
2000	0.0155
3000	0.0160
5000	0.0143
10000	0.0147
20000	0.0161
30000	0.0173
50000	0.0202


binomial heap union
0.025
0.02
0.015
0.01
0.005
0
time in millisec
0
10000
20000
30000
40000
50000
input size