Practical 3
Implementation of Binomial Heap

Gahan M. Saraiya, 18MCEC10

18mcec10@nirmauni.ac.in

I. Introduction

Aim of this practical is to analyze the time complexity of sorting algorithms for various input size.

Algorithms analyzed are listed below:

II. Implementation

I. Utility *utility.h*

```
//
// Created by jarvis on 17/8/18.
//

#ifndef DSA_LAB_UTILITY_H
#define DSA_LAB_UTILITY_H

#include <string.h>
#include <stdarg.h>

int write_log(const char *format, ...) {
    if(DEBUG) {
        printf("\n[DEBUG_LOG]> ");
        va_list args;
        va_start (args, format);
        vprintf(format, args);
        va_end (args);
    }
}

int *get_min_max(int *array, int no_of_elements, int min_max[]){
    // get minimum and maximum of array
//    printf("elements of array: ");
    for(int i=0; i<no_of_elements; i++){
//        printf("%d ", *(array + i));
        if (*(array + i) < min_max[0])
            min_max[0] = *(array + i);
        if (*(array + i) > min_max[1])
```

```
29              min_max[1] = *(array + i);
30      }
31      return min_max;
32  }
33
34  int display_array(int *array, int no_of_elements){
35      // display given array of given size(no. of elements require because sizeof()
        ↪  returns max bound value)
36      write_log(": ");
37      for(int i=0; i<no_of_elements; i++){
38          write_log( "%d ", *(array + i));
39      }
40      return 0;
41  }
42
43  int show_2d_array(int array[2048][2048], int no_of_elements){
44      // display given array of given size(no. of elements require because sizeof()
        ↪  returns max bound value)
45      write_log(": ");
46      for(int i=0; i<no_of_elements; i++){
47          printf("a[%d][]: ", i);
48          for(int j=0; j<no_of_elements; j++) {
49  //              printf("array[%d][%d]: %d ", i, j, array[i][j]);
50              printf("%d ", array[i][j]);
51          }
52          printf("\n");
53      }
54      return 0;
55  }
56
57  int display_2d_array(int **array, int no_of_elements){
58      // display given array of given size(no. of elements require because sizeof()
        ↪  returns max bound value)
59      write_log(": ");
60      for(int i=0; i<no_of_elements; i++){
61          printf("a[%d][]: ", i);
62          for(int j=0; j<no_of_elements; j++) {
63  //              printf("array[%d][%d]: %d ", i, j, array[i][j]);
64              printf("%d ", array[i][j]);
65          }
66          printf("\n");
67      }
68      return 0;
69  }
70
71
72  void swap(int *one, int *two){
73      // swap function to swap elements by location/address
```

```
74      int temp = *one;
75      *one = *two;
76      *two = temp;
77  }
78
79  #endif //DSA_LAB_UTILITY_H
```

II. Main Program - *binomial_heap.c*

```
1   //
2   //
    ↪ ----------------------------------------------------------------------------------------
3   // Author: Gahan Saraiya
4   // GiT: http://github.com/gahan9/
5   // StackOverflow: https://stackoverflow.com/users/story/7664524
6   // Website: http://gahan9.github.io/
7   //
    ↪ ----------------------------------------------------------------------------------------
8   // Implementing Binomial Heap
9
10  #include <stdio.h>
11  #include <stdlib.h>
12  #include <malloc.h>
13
14  #define DEBUG  1
15
16  #include "../utils/utility.h"
17
18  struct node {
19      // Node for binomial heap
20      int n;
21      int degree;
22      struct node *parent;
23      struct node *child;
24      struct node *sibling;
25  };
26
27  int count = 1;
28
29  struct node *MAKE_bin_HEAP() {
30      struct node *np;
31      np = NULL;
32      return np;
33  }
34
35  struct node *H = NULL;
36  struct node *Hr = NULL;
```

```
37
38  int linkBinomialHeap(struct node *y, struct node *z) {
39      y->parent = z;
40      y->sibling = z->child;
41      z->child = y;
42      z->degree = z->degree + 1;
43  }
44
45  struct node *CreateNode(int k) {
46      struct node *p;//new node;
47      p = (struct node *) malloc(sizeof(struct node));
48      p->n = k;
49      return p;
50  }
51
52  struct node *BinomialHeapMerge(struct node *H1, struct node *H2) {
53      struct node *H = MAKE_bin_HEAP();
54      struct node *y;
55      struct node *z;
56      struct node *a;
57      struct node *b;
58      y = H1;
59      z = H2;
60      if (y != NULL) {
61          if (z != NULL && y->degree <= z->degree)
62              H = y;
63          else if (z != NULL && y->degree > z->degree)
64              /* need some modifications here;the first and the else conditions can
                ↪ be merged together!!!! */
65              H = z;
66          else
67              H = y;
68      } else
69          H = z;
70      while (y != NULL && z != NULL) {
71          if (y->degree < z->degree) {
72              y = y->sibling;
73          } else if (y->degree == z->degree) {
74              a = y->sibling;
75              y->sibling = z;
76              y = a;
77          } else {
78              b = z->sibling;
79              z->sibling = y;
80              z = b;
81          }
82      }
83      return H;
```

```
}

struct node *BinomialHeapUnion(struct node *H1, struct node *H2) {
    struct node *prev_x;
    struct node *next_x;
    struct node *x;
    struct node *H = MAKE_bin_HEAP();
    H = BinomialHeapMerge(H1, H2);
    if (H == NULL)
        return H;
    prev_x = NULL;
    x = H;
    next_x = x->sibling;
    while (next_x != NULL) {
        if ((x->degree != next_x->degree) || ((next_x->sibling != NULL)
                                              && (next_x->sibling)->degree ==
                                               ↪  x->degree)) {
            prev_x = x;
            x = next_x;
        } else {
            if (x->n <= next_x->n) {
                x->sibling = next_x->sibling;
                linkBinomialHeap(next_x, x);
            } else {
                if (prev_x == NULL)
                    H = next_x;
                else
                    prev_x->sibling = next_x;
                linkBinomialHeap(x, next_x);
                x = next_x;
            }
        }
        next_x = x->sibling;
    }
    return H;
}

struct node *BinomialHeapInsert(struct node *H, struct node *x) {
    struct node *H1 = MAKE_bin_HEAP();
    x->parent = NULL;
    x->child = NULL;
    x->sibling = NULL;
    x->degree = 0;
    H1 = x;
    H = BinomialHeapUnion(H, H1);
    return H;
}

```

```
131 //int NodeExplorer(struct node *n){
132 //    write_log("Exploring node: %d\n", n->n);
133 //    if (n == NULL){
134 //        printf("Empty Node!!!\n");
135 //        return 0;
136 //    }
137 //    else{
138 //        printf("%d, ", n->n);
139 //        if (p->sibling != NULL)
140 //            printf("\n-->");
141 //    }
142 //    printf("\n");
143 //}
144
145 int DisplayBinomialHeap(struct node *H) {
146     // TODO: Remove redundant print... do pretty print of heap
147 //    printf("\n");
148     struct node *p;
149     if (H == NULL) {
150         printf("\nHEAP EMPTY");
151         return 0;
152     }
153 //    printf("\nTHE HEAP:-\n");
154     p = H;
155     while (p != NULL) {
156         printf("[%d]", p->n);
157         if (p->sibling != NULL) {
158 //            printf("Parent: %d", p->parent->n);
159             printf("------------>");
160             DisplayBinomialHeap(p->sibling);
161         }
162
163         if (p->child != NULL){
164             printf("\nParent: of %d is %d\n",p->child->n, p->n);
165             DisplayBinomialHeap(p->child);
166         }
167         else if (p->child == NULL){
168             printf("---[leaf node] Nothing to explore for this node.");
169         }
170         p = p->sibling;
171     }
172     printf("\n");
173     return 1;
174 }
175
176 int RevertList(struct node *y) {
177     if (y->sibling != NULL) {
178         RevertList(y->sibling);
```

```
179         (y->sibling)->sibling = y;
180     } else {
181         Hr = y;
182     }
183 }
184
185 struct node *ExtractMinBinomialHeap(struct node *H1) {
186     int min;
187     struct node *t = NULL;
188     struct node *x = H1;
189     struct node *Hr;
190     struct node *p;
191     Hr = NULL;
192     if (x == NULL) {
193         printf("\nNOTHING TO EXTRACT");
194         return x;
195     }
196     //    int min=x->n;
197     p = x;
198     while (p->sibling != NULL) {
199         if ((p->sibling)->n < min) {
200             min = (p->sibling)->n;
201             t = p;
202             x = p->sibling;
203         }
204         p = p->sibling;
205     }
206     if (t == NULL && x->sibling == NULL)
207         H1 = NULL;
208     else if (t == NULL)
209         H1 = x->sibling;
210     else if (t->sibling == NULL)
211         t = NULL;
212     else
213         t->sibling = x->sibling;
214     if (x->child != NULL) {
215         RevertList(x->child);
216         (x->child)->sibling = NULL;
217     }
218     H = BinomialHeapUnion(H1, Hr);
219     return x;
220 }
221
222 struct node *FIND_NODE(struct node *H, int k) {
223     struct node *x = H;
224     struct node *p = NULL;
225     if (x->n == k) {
226         p = x;
```

```
227         return p;
228     }
229     if (x->child != NULL && p == NULL) {
230         p = FIND_NODE(x->child, k);
231     }
232 
233     if (x->sibling != NULL && p == NULL) {
234         p = FIND_NODE(x->sibling, k);
235     }
236     return p;
237 }
238 
239 int bin_HEAP_DECREASE_KEY(struct node *H, int i, int k) {
240     int temp;
241     struct node *p;
242     struct node *y;
243     struct node *z;
244     p = FIND_NODE(H, i);
245     if (p == NULL) {
246         printf("\nINVALID CHOICE OF KEY TO BE REDUCED");
247         return 0;
248     }
249     if (k > p->n) {
250         printf("\nSORY!THE NEW KEY IS GREATER THAN CURRENT ONE");
251         return 0;
252     }
253     p->n = k;
254     y = p;
255     z = p->parent;
256     while (z != NULL && y->n < z->n) {
257         temp = y->n;
258         y->n = z->n;
259         z->n = temp;
260         y = z;
261         z = z->parent;
262     }
263     printf("\nKEY REDUCED SUCCESSFULLY!");
264 }
265 
266 int bin_HEAP_DELETE(struct node *H, int k) {
267     struct node *np;
268     if (H == NULL) {
269         printf("\nHEAP EMPTY");
270         return 0;
271     }
272 
273     bin_HEAP_DECREASE_KEY(H, k, -1000);
274     np = ExtractMinBinomialHeap(H);
```

```
275     if (np != NULL)
276         printf("\nNODE DELETED SUCCESSFULLY");
277 }
278 
279 int main(int argc, char *argv[]) {
280     int i, n, m, l;
281     struct node *p;
282     struct node *np;
283     char ch;
284     int number_of_elements;
285     if (atoi(argv[1]))
286         number_of_elements = atoi(argv[1]);
287     else
288         number_of_elements = 5;  // elements to be pre filled
289     for (i = 1; i <= number_of_elements; i++) {
290         m = rand() % 10;
291         printf("Inserting: %d\n", m);
292         np = CreateNode(m);
293         H = BinomialHeapInsert(H, np);
294     }
295 //    DisplayBinomialHeap(H);
296     do {
297         printf("\n#####################***MENU***############################"
298                "\n1. Insert"
299                "\n2. Extract Minimum key Node"
300                "\n3. Display Binomial Heap"
301                "\n4. Exit"
302                "\n#############################################################\n");
303         scanf("%d", &l);
304         switch (l) {
305             case 1:
306                 printf("\n[INSERT]:");
307                 scanf("%d", &m);
308                 write_log("Inserting: %d", m);
309                 p = CreateNode(m);
310                 H = BinomialHeapInsert(H, p);
311                 printf("\n<<<HEAP>>>");
312                 DisplayBinomialHeap(H);
313                 scanf("%c", &ch);
314                 break;
315             case 2:
316                 printf("\nExtracting min key node");
317                 p = ExtractMinBinomialHeap(H);
318                 if (p != NULL)
319                     printf("\nExtracted node is %d", p->n);
320                 printf("\n<<<HEAP>>>");
321                 DisplayBinomialHeap(H);
322                 scanf("%c", &ch);
```

```
                break;
            case 3:
                printf("\n<<<HEAP>>>");
                DisplayBinomialHeap(H);
                break;
            case 4:
                printf("\nGratitude! for interacting with program\n");
                break;
            default:
                printf("\nOoops.... that's invalid choice\n");
                break;
        }
    } while (l != 4);
}
```

III. Output

```
Inserting: 3
Inserting: 6
Inserting: 7
Inserting: 5
Inserting: 3

#####################MENU############################
1. Insert
2. Extract Minimum key Node
3. Display Binomial Heap
4. Exit
#####################################################

<<<HEAP>>>[3]------------>[3]
Parent: of 5 is 3
[5]------------>[6]---[leaf node] Nothing to explore for this node.

Parent: of 7 is 5
[7]---[leaf node] Nothing to explore for this node.
[6]---[leaf node] Nothing to explore for this node.

---[leaf node] Nothing to explore for this node.[3]
Parent: of 5 is 3
[5]------------>[6]---[leaf node] Nothing to explore for this node.

Parent: of 7 is 5
```

```
[7]---[leaf node] Nothing to explore for this node.
[6]---[leaf node] Nothing to explore for this node.



######################MENU#############################
1. Insert
2. Extract Minimum key Node
3. Display Binomial Heap
4. Exit
########################################################

Gratitude! for interacting with program
```