Algorithms

Assignment 2

Abdul Sami Qasim
22i-1725
CY-D

To :-

Mrs. Amna Siddique

Q1) merge vs quick

1) Merge sort has better time complexity O(nlogn)

2) merge can be performed via parallel processing

3) merge is stable

4) merge works well on large arrays

b)

worst case is when pivot is unbalanced leading to 1 element on one side and n-1 on the other.

c)

Input is already sorted and it is best case

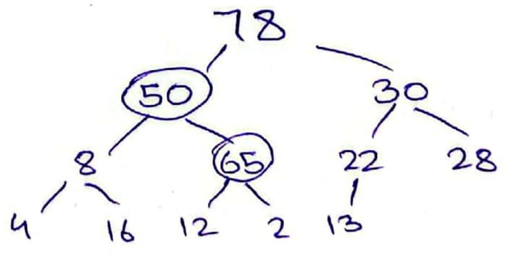
d)

Insertion sort is best

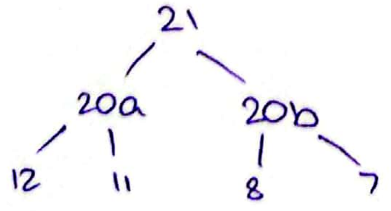
Q2) a)

This is not a max-heap

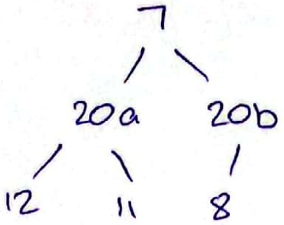
fails on 50 and 65

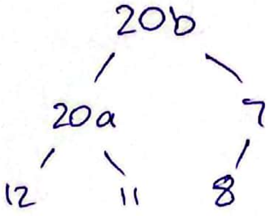


b)



lets delete 21,





at this step, we can see that heapsort fails at being stable.

c)

$$T(n) = \sum_{i=0}^{h} n_i h_i$$

$$= \sum_{i=0}^{h} 2i(h-i)$$

$$= \sum_{i=0}^{h} \frac{h-i}{2^{h-i}} 2^h \qquad (k=h-i)$$

$$= 2^h \sum_{k=0}^{h} \frac{k}{2^k} \leq n \sum_{k=0}^{\infty} \frac{k}{2^k}$$

$$= O(n)$$

Q3) a)

```
swapends (s){

    v1 = s.removefromfront();
    v2 = s.removefromback ();
    s. addtofront (v2);
    s. addtoback (v1);
  }
```

b)

```
Shiftleft(s,k){
        for (i=1 to k){
                v1 = s.removefromfront();
                s.addtoback(v1);
        }
}
```

Q4)a)

minheap to keep track of top scores k, the smallest score is slowly removed. This is done in time $O(|A|+k\log|A|)$

b)

using max-heap with a threshold value of x would be best in this case. all the gardens that have a score $\geq x$ will be rewarded.

Q5)
1)

The best case for naive is if the pattern is right at the start or if the pattern is very unique and many mismatches have not been made.

Worst case is if too many mismatches have been made.

best case example :-
pattern = cd
string = cdabab

worst case example :-

pattern :- ab
string :- aaaaacaaaab

2) I suggest learning how many positions you can skip based on previous mismatches to keep comparisons at the lowest. Although this is not feasible because better algorithms like KMP and Rabin-Karp already exist.

Code :-

```
naiveplusplus( string t, string p){
        int n = t.size();
        int m = p.size();
        for (int i=0; i<=n-m; i++){
                int j=0;
                while ( j<m && ~~pattern[j]~~ p[j]== t[i+j]){
                        j++;
                }
                if ( j==m){
                        cout<<" pattern is at:"<<i <<endl;
                        i+=m;
                }
                else{
                        if(j==0){
                                i++;
                        }
                        else{
                                i+=j;
                        }
                }
        }
}
```

These if-else and the nested if-else are responsible for optimizing the code as they skip places if mismatches occur and save time.

Q6) a)

a	a	b	a	a	b	c	a	b
0	1	0	1	2	3	0	1	2

prefix function gives us:-

0, 1, 0, 1, 2, 3, 0, 1, 2

b)

```
prefixcalc ( string p) {
        int  m = p.size();
        pi [0] = 0;
        int  k = 0;
        for ( int  q=i ; q<m; q++){
                while ( k>0  &&  p[k] != p [q] ){
                        k++;
                }
                pi [q] = k ;
        }
}
```

Q7) a)

$p = 26$

$q = 11$

Calculating hash for all the string:-

31	=	9
14	=	3
41	=	8
15	=	4
59	=	4
92	=	4
26	=	4
65	=	10
53	=	9
35	=	2
58	=	3
89	=	1
97	=	9
79	=	2
93	=	5

There are 3 spurious hits in this question

b)

```
karp( char t[], char p[], int d, int q){
        int  n= t.size();
        int  m= p.size();
        int  h= 1;
        int  p= 0;
        int  t1= 0;

        for ( int i=0; i<m-1; i++){
                h= (h * d) % q;
        }
        for ( int i=0; i<m; i++){
                p= d * p + p[i] % q;
                t1= d * t1 + t[i] % q;

        }
for (int s=0; s≤n-m; s++){
    if (p==t1){
        bool match= true;
        for (i=0; i<m; i++){
                if (t [s+i] != p[i]){
                        match=false;
                        break;
                }
        }
        if (match){
                cout<<"pattern is at: " << s <<endl;
        }
    }
    if (s≤n-m){
        t1=(d * (t1 - t[s] * h) + t[s+m]) % q;
        if (t1<0){
                t1 = t1+q;
        }
    }
}
}
```

Q8) If we have to search for ~~an~~ nxn pattern in a mxm array, we will calculate the hash of nxn blocks (smaller than mxm) to match with the given pattern.

Q9) a)

Bayer-Moore algorithm:-

This makes a shift table in preprocessing in order to see how many shifts can be made upon a mismatch.

example:-

```
ABAABABCBBABDBAB
ABD
```

last element doesnt match, go to next occurence of A

```
ABAABABCBBABDBAB
  ABD
```

```
ABAABABCBBABDBAB
   ABD
```

```
ABAABABCBBABDBAB
     ABD
```

```
ABAABABCBBABDBAB
          ABD
```

This is matched and no further checking can be done.

b)

```
table( string s, int size, int arr[256]) {
   int i;
 for (int ~~{~~ i=0;i<256;i++){
          arr [i] = -1;
   }

 for (i=0; i<size ;i++){
      arr[(int) s[i]] = i;

 }
}
```

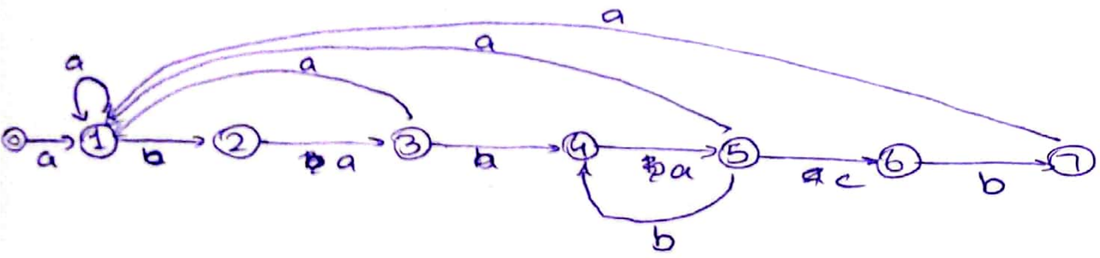
```
search ( string text, string pattern) {
        int  m = pattern.size();
        int  n = text.size();
        int arr [256];
        shift (pattern,m,arr);
        int  s = 0;
        while (s ≤ (n-m)) {
                int  j = m-1;
                while (j≥0 && pattern[j] == text [s+j]) {
                        j--;
                }
                if (j<0){
                        cout<<" pattern at: " << s <<endl;
                        s+=(s+m <n)? m- arr [text[s+m]] : 1;
                }
                else{
                        s+= max (1, j- arr [text[s+j]]);
                }
        }
}
```

c) 1)

string = A = abaababababacb

pattern = ababacb

state	a	b	c
0	1	0	0
1	1	2	0
2	3	0	0
3	1	4	0
4	5	0	0
5	1	4	6
6	1	7	0
7	1	0	0

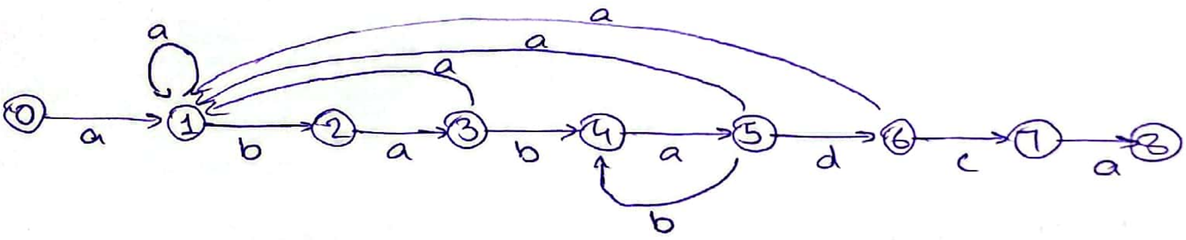


A	state
a	1
b	2
a	3
a	1
b	2
a	3
b	4
a	5
b	4
a	5
b	4
a	5
c	6
b	7

2)

S= ababadca

state	a	b	c	d
0	1	0	0	0
1	1	2	0	0
2	3	0	0	0
3	1	4	0	0
4	5	0	0	0
5	1	4	0	6
6	1	0	7	0
7	8	0	0	0



i) A= abababdcaababadca

A	state
a	1
b	2
a	3
b	4
a	5
b	4
d	0
c	0
a	1
a	1
b	2
a	3
b	4
a	5
d	6
c	7
a	8

ii) A= ababadcaababadcad

A	state
a	1
b	2
a	3
b	4
a	5
d	6
c	7
a	⑧
a	1
b	2
a	3
b	4
a	5
d	6
c	7
a	⑧
d	0

iii) ~~ab~~-A= abababdcbababadcb

A	state
a	1
b	2
a	3
b	4
a	5
b	4
d	0
c	0
b	0
a	1
b	2
a	3
b	4
a	5
d	6
c	7
b	0

S is substring in i and ii

S is not found in iii