Practice

This document has some question and answers in order to test knowledge on the subject of TOA, please attempt the question before looking at the answer.

2017 Theory Of Algorithms

Question 1

1. Briefly explain the techniques below. Be sure the difference between the two problem-solving strategies in each pair is clear. (6)
    1. Divide-and-conquer and decrease-and-conquer
    2. Space-and-time tradeoff and problem reduction
    3. Dynamic programming (DP) and DP with memory functions

▼ View answer 1.1
Divide and conquer is to solve a problem by combining solutions of smaller subproblems.

Decrease-conquer means to solve a problem by solving smaller instance of that same problem

▼ View answer 1.2
Space-and-time tradeoff is when you solve a problem faster by using more space (or vice versa, if space is the problem).

Problem reduction is when you solve problem A by transforming it into another problem B.

▼ View answer 1.3
DP is when you solve a problem by combining solutions of overlapping smaller subproblems.

DP with memory functions is as above, but proceed top down instead of bottom up

2. State whether each of the following is true or false:

▼ $\Theta (n + \log n) = \Theta(n)$
True
▼ $O (n \log n) = O(n)$
False
▼ if $p \in O (n \log n)$ then $p \in O (n^2)$
True
▼ if $p \in \Theta (n \log n)$ then $p \in \Theta (n^2)$
False
▼ if $p \in \Omega (n \log n)$ then $p \in \Omega (n)$
True

3. Exponentiation function a^n calculates a to the power n. Show how the problem of find a^n is tackled using each of the following techniques:
    1. Brute Force
    2. Decrease by a constant

3. Decrease by a constant factor
4. Divide & Conquer

▼ Brute Force

a * a * a * ... * a (n times)

▼ Decrease by a constant

$a^{n-1}*a$

▼ Decrease by a constant factor

$(a^{n/2})^2$

▼ Divide & Conquer

$a^{n/2} * a^{n/2}$

4. Use the Master Theorem to find the complexity of the function below, show your working, and remember that the Master Theorem is as follows:

if T(n) = aT(n/b) + f(n) and f(n) ∈ Θ (n^2) then:

- T(n) ∈ Θ(n^d) if a < b^d
- T(n) ∈ Θ(n^dlogn) if a = b^d
- T(n) ∈ Θ($n^{\log_b a}$) if a > b^d

The function looks like this:

```
waat(x):
    if x < 3:
        return a


    else:
         return waat (2x/3) ++ waat(2x/3)
```

▼ Answer

If we look at the function, we can put it into the form of the Master Theorem. We can see that: a = 2, b = 3/2 and that d is equal to 0. Thus if we put these into the equation a = b^d it will be equal to: 2 = $3/2^0$. Thus a > b^d. Therefore, T(n) ∈ Θ($n^{\log_{3/2}2}$)

Question 2

Consider searching for the first occurrence of the octal pattern 061606 in octal text 02040606061606

1. Show the shift table that would be used if Horspool's algorithm is applied (2)
2. Show the good-suffix table that Boyer-Moore would use for this search (4)
3. The search string starts by aligning the pattern with the first character of the text. Where will it move the pattern to next using:
    1. Horspool
    2. Boyer-Moore
4. Can the Boyer-Moore algorithm ever have a good-suffix shift table with elements S[j] and S[k] such that j < k but S[j] > S[k]? If no, explain why not. If yes, give an example of such a search pattern. (1)

5. Which algorithm is more efficient: Horspool or Boyer-Moore? (1)

▼ 2.1

The shift table used for Horspool can be represented as follows:

0	**6**	**1**	*****
1	2	3	6

Where * represents all other characters in the alphabet, and the values are the position of the character furthest to the right in regard to the last character.

▼ 2.2

The good suffix table can be represented as follows:

k	**Pattern**	**shift**
1	061**6**0**6**	2
2	**06**16**06**	4
3	061**606**	4
4	06**1606**	4
5	0**61606**	4

▼ 2.3

1. Horspool shifts up 2 places
2. Boyer-Moore shifts up 4 places

▼ 2.4

Yes, many example such as: anon, teepee, ceded.

▼ 2.5

Boyer-More is more efficient

Horspool Algorithm Example

Example: Consider searching for the pattern BARBER in the text below: `JIM_SAW_ME_IN_A_BARBERSHOP`.

▼ Answer

First construct the shift table for BARBER, there are 4 unique letters in BARBER - every other letter will receive a value of 6 as that is the total length of BARBER. Thus the shift pattern will be:

B	**A**	**R**	**E**	*****
2	4	3	1	6

Where * represents all other characters, and the value in the table is the index of the last character of that character in the pattern BARBER.

Now we have our shift table we can start going through the text.

```
JIM_SAW_ME_IN_A_BARBERSHOP
BARBER
```

So, R does not match with A, but it is in our shift table - so we can move our pattern by 4.

```
JIM_SAW_ME_IN_A_BARBERSHOP
    BARBER
```

Again, R does not match E, but E is in our shift table to move by 1.

```
JIM_SAW_ME_IN_A_BARBERSHOP
     BARBER
```

The E, does match now thanks to our shift table - but the R is still in the wrong location it is on a space, which is not in our pattern - so we can move the entire pattern by 6.

```
JIM_SAW_ME_IN_A_BARBERSHOP
           BARBER
```

The B does not match the R in our pattern. But we do have a rule for B which is to move by 2.

```
JIM_SAW_ME_IN_A_BARBERSHOP
             BARBER
```

The The R does match now, but the A does not - so we have to move our pattern by 3

```
JIM_SAW_ME_IN_A_BARBERSHOP
                BARBER
```

Now all our characters in the pattern match! We have found a match. This is all of the steps we took in one snippet.

```
JIM_SAW_ME_IN_A_BARBERSHOP
BARBER      BARBER
    BARBER    BARBER
     BARBER      BARBER
```

Boyer Moore Algorithm Example

As a example, consider searching for the pattern BAOBAB in the text `BESS_KNEW_ABOUT_BAOBABS`.

▼ Answer

So to start we need to create our bad shift table, and our good shift table. The bad shift table takes the word BAOBAB and uses the unique characters to create shifting rules - exactly like in Horspool.

B	A	O	*
2	1	3	6

During the pattern matching, we change the values based on the bad-symbol shift which is the function `max(t(c)-k,1)` - where `t(c)` is the value in the shift table for the mismatched character, and `k` is the number of matched characters before that was hit.

Then, we can create the good suffix table for the word BAOBAB

k	pattern	shift
1	BAOBA**B**	2
2	BAOB**AB**	5
3	BAO**BAB**	5
4	BA**OBAB**	5
5	B**AOBAB**	5

As you can see, the suffix fails to match after *suff(1)*

Now that we have the two tables we can start applying them on the text:

```
BESS_KNEW_ABOUT_BAOBABS
BAOBAB
```

The value for `K` (for `KNEW`)in the the shift table is 6 the number of matched characters (k) is 0. Thus, the equation for the shift table is now: max(6-0,1) - which is obviously 6. Thus we shift the entire pattern 6 places.

```
BESS_KNEW_ABOUT_BAOBABS
      BAOBAB
```

Now, both B and A are matching. Forming the suffix `AB`, so at this point we need to check which is better - using the good suffix table or the bad shift table. For the suffix `AB` the shift value is 5. Then the value for the mismatched char (the _)in our bad shift table is 6, but we must reduce that by the number of matched character (k). Thus our equation will be `max(5,max(6-2,1))` which equals 5 (as 5 is greater than 4). Thus we will shift our pattern by 5.

```
BESS_KNEW_ABOUT_BAOBABS
           BAOBAB
```

B matched with B - so we can do the same as above. Compare the value in our good suffix table, and our bad shift table and take the maximum of both. Hence, we will have `max(2,max(6-1,1))` = 5. Thus we will shift our pattern by 5 places.

```
BESS_KNEW_ABOUT_BAOBABS
                    BAOBAB
```

And there we have it, we have matched the pattern to the text. Again, here are our iterations in one case:

```
BESS_KNEW_ABOUT_BAOBABS
BAOBAB
        BAOBAB
              BAOBAB
                    BAOBAB
```