Mohammad Hoque
6265549
1/21/19

HW 1

① 1. $f_1(n), f_2(n), f_4(n), f_3(n)$

$f_1(n) = n^{\cdot 99 \log n} = O(n^{\log n} \cdot n^{\log n}) = O(n) = O(f_3(n))$

$f_4(n) = \text{recursive} \Rightarrow f_2(n) = f_4(n)$

$f_3(n) \Rightarrow \text{exponential} \Rightarrow f_3(n) = O(f_3(n))$

2. $f_1(n), f_3(n), f_4(n), f_2(n)$

$f_1(n) \Rightarrow \text{constant}$; $f_3(n) = n \log n = O(n \log n)$

$f_4(n) = n^{1.5} = O(n^2)$

$f_2(n) \Rightarrow \text{exponential}$

3. $f_4(n), f_1(n), f_3(n), f_2(n)$

$f_4(n) = \frac{n(n+1)+2}{2} = \frac{n(n+3)}{2} = \Theta(n^2) < f_1(n)$

$f_1(n) = n^{\sqrt{n}} = (2^{\log n})^{\sqrt{n}} = 2^{\sqrt{n} \log n}$

$f_3(n) = n^{10} \cdot 2^{n/2} = 2^{\log(n^{10})} \cdot 2^{n/2} = 2^{n/2 + 10 \log n}$ $\Big\} f_1(n) = O(f_3(n))$

$f_3(n) = O(f_2(n))$

② a) $a = 2, b = 3, f(n) = n \log n$

Case 3 of Master Method $\Rightarrow T(n) = \Theta(n \log n)$

b) $a = 3, b = 5, f(n) = \log^2 n$

Case 1 of Master Method $\Rightarrow T(n) = \Theta(n^{\log_5 3})$

c) $a = 1, b = 2, f(n) = 2^n$

Case 3 $\Rightarrow \Theta(2^n)$

d) let $n = 2^m \Rightarrow S(m) = S(m/2) + \Theta(\log m)$

Case 2 $\Rightarrow T(n) = \Theta((\log \log n)^2)$

e) $\log_3 10 > 2 > 1.2$; $a = 10, b = 3, f(n) = 17 n^{1.2}$

Case 1 $\Rightarrow T(n) = \Theta(n^{\log_3 10})$

f) $f(n) = n^3$

Case 3 $\Rightarrow T(n) = \Theta(n^3)$

g) $T(n/2) \le T(n/2 + \sqrt{n}) \le T(3n/4)$

$\Rightarrow T(n) = \Theta(\log n)$

h) $T(n) = \sum_{i=1}^{n/2} \log i \ge \sum_{i=n/4}^{n/2} \ge (n/4)(\log n/4) = \Omega(n \log n)$

Upper bound: $T(n) \le S(n) = S(n-1) + \log n = O(n \log n)$

$T(n) = \Theta(n \log n)$

i) $\log_{5/4} n = \Theta(\lg n) \Rightarrow T(n) = \Theta(n \lg n)$; Substitution method

$f(n) = \Theta(n) \Rightarrow$ for all $n \geq n_0$, $c_1 n \leq f(n) \leq c_2 n$

$T(n) = O(n)$: $T(n) \leq d_1 n \lg n$

inductive step: for all $k < n$, $T(k) \leq d_1 k \lg k$

for $k = n$: $T(n) \leq T(\frac{n}{5}) + T(\frac{4n}{5}) + c_2 n \leq d_1 \frac{n}{5} \lg(\frac{n}{5}) + d_1 \frac{4n}{5} \lg(\frac{4n}{5}) + c_2 n = d_1 n \lg n - n\left(\left(\frac{\lg 5 + 4\lg(5/4)}{5}\right) d_1 - c_2\right)$

$\Rightarrow T(n) = O(n \lg n)$

$T(n) = \Omega(n)$: $T(n) \geq d_0 n \lg n$

inductive step: for all $k < n$, $T(k) \geq d_0 k \lg k$

for $k = n$: $T(n) \geq d_0 \frac{n}{5} \lg(\frac{n}{5}) + d_0 \frac{4n}{5} \lg(\frac{4n}{5}) + c_1 n = d_0 n \lg n + n\left(c_1 - \left(\frac{\lg 5 + 4\lg(5/4)}{5}\right) d_0\right)$

$\Rightarrow T(n) = \Omega(n \lg n)$

ii) $S(n) = \frac{T(n)}{n} \Rightarrow S(n) = S(\sqrt{n}) + c_3$

let $m = \lg n \Rightarrow S(m) = S(\frac{m}{2}) + c_3$

$T(n) = \Theta(n \lg \lg n)$

③ A.2.3-5: Binary search searches an array by halving the search interval. If value of search key is less the item in the middle, narrow the interval to the lower half, otherwise the upper. Repeat until value is found or empty

Iterative pseudocode:

```
BinarySearch[A,V]{
  begin = 0;
  last = A.length - 1;
  While (begin ≤ last){
    mid = (begin + last)/2;
    if V == A(mid)
      return mid;
    else if V > A(mid)
      last = mid + 1;
    else
      begin = mid - 1
  return NIL
```

worst case would be when V is not present in A and search returns NIL after searching entire array. Running time is how many times the input is halved, $\lg n$.

$\Rightarrow T(n) = T((n-1)/2) + \Theta(1) = \Theta(\lg n)$

(3) A: 2.3-6:

Using binary search we still have to shift elements to the end of the array which runs at $\Theta(n)$ time for average case. For the worst case the running time stays the same, thus it cannot be improved to $\Theta(n \lg n)$ using binary search.

B. Average execution times (in seconds)

n	insertion	merge	n	insertion	merge
2^0	∅	∅	2^{14}	.0292	.0015
2^1	∅	1e-06	2^{15}	.1164	.0031
2^2	∅	∅	2^{16}	.4841	.0065
2^3	∅	2e-06	2^{17}	2.0003	.0138
2^4	1e-06	1e-06	2^{18}	8.2112	.0289
2^5	2e-06	3e-06	2^{19}	32.8457	.0610
2^6	2e-06	4e-06	2^{20}	131.198	.1280
2^7	7e-06	7e-05	2^{21}		.2693
2^8	1.4e-05	1.6e-05	2^{22}		Seg Fault
2^9	5.4e-05	3.3e-05	2^{23}		
2^{10}	.0018	7e-05	2^{24}		
2^{11}	.0053	.0016	2^{25}		
2^{12}	.0021	.0039	2^{26}		
2^{13}	.0076	.0068			

mergesort's runtime fluctuated in the beginning and was slightly slower than insertion but then progressively became faster but eventually seg faulted whereas insertion sort kept going.