Design and Analysis of Algorithms

4th Semester, CST Department

Theory Paper

Complexity – asymptotic bounds

- An approximate measure of the time taken for the program to run
- Stated as a function f(n) of the problem size
- Problem size for sorting – size of the array
- Factorization of one integer – its magnitude
- Inequality in terms of known function g(n)

$$c_1\, g(n) \leq f(n) \leq c_2\, g(n) \text{ for all } n > n_0$$

Complexity – asymptotic bounds

- Tight bound Big theta ***f(n) = Θ (g(n))***

$$c_1\, g(n) \leq f(n) \leq c_2\, g(n)$$

- Tight upper bound Big-O ***f(n) = O (g(n))***

$$f(n) \leq c_2\, g(n)$$

- Tight lower bound Big Omega ***f(n) = Ω (g(n))***

$$c_1\, g(n) \leq f(n)$$

- Upper bound small-o ***f(n) = o (g(n))***

$$f(n) < c_2\, g(n)$$

- Lower bound small-omega ***f(n) = ω (g(n))***

$$c_1\, g(n) < f(n)$$

Outline Syllabus – different paradigms

- Divide and conquer approach
  - Sorting algorithms
- Dynamic programming approach
  - Matrix multiplication
- Greedy strategy
  - Minimum spanning tree in graph
- Backtracking strategy
  - Depth first and breadth first search in graph

Outline syllabus - some data structures and operations

- Operation on Disjoint sets
  - Shortest path in graph
- Operation on dynamic sets
  - Hashing
- Operation on Polynomials
  - FFT algorithm
- Operations in number theory
  - RSA algorithm

Outline syllabus – advanced topics

- Complexity theory
- Randomized algorithms
- Parallel algorithms
- NP completeness
- Approximation algorithms

Divide and conquer strategy

- DIVIDE into subproblems
- CONQUER the subproblems solving recursively
- COMBINE the solutions of subproblems

Consider a subproblems, each of size (1/b)

Time to divide = D(n), Time to combine = C(n)

$T(n) = a\,T(n/b)+D(n)+C(n)$

Analogy in Merge Sort

- DIVIDE: n-element sequence into n/2 elements in two subsequences
- CONQUER: sort the two subsequences recursively
- COMBINE: MERGE the two subsequences

$D(n) = \Theta(1)$, $C(n) = \Theta(n)$, $a=2$ and $b=2$

$T(n) = 2\,T(n/2)+\Theta(n)$ giving $T(n) = \Theta(n \lg n)$

MERGE SORT algorithm

```
MERGE_SORT (A,p,r)
    if (p<r)
        q = floor((p+r)/2)
        MERGE_SORT(A,p,q)
        MERGE_SORT(A,q+1,r)
        MERGE(A,p,q,r)
```

Initial sequence: (5);(7);(6);(4);(1);(3);(2);(8)

Step 1 of conquer: (5,7) ; (4,6); (1,3); (2,8)

Step 2 of conquer: (4,5,6,7) ; (1,2,3,8)

Step 3 of conquer: (1,2,3,4,5,6,7,8)

Quick Sort Algorithm

```
QUICK_SORT(A,p,r)
    if (p<r)
        q=PARTITION(A,p,r)
        QUICK_SORT(A,p,q)
        QUICK_SORT(A,q+1,r)
PARTITION(A,p,r)
    x=A[p]
    i=p-1, j=r+1
    while TRUE
        repeat j=j-1 until A[j] ≤ x
        repeat i=i+1 until A[i] ≥ x
If i<j
    exchange A[i], A[j]
Else
    return (j)
```

Solving recurrence

- Substitution method
  - Guess a bound and then use mathematical induction to prove the guess correct
- Iteration method
  - Convert the recurrence into a summation and then bound the summation
- Master method
  - Provides bound for all recurrences of the form $T(n) = a\,T(n/b) + f(n)$

Substitution method

- *T(n)=2 T(n/2) + n* → Guess *T(n) = O(n lg n)*
- Start by assuming this bound holds for T(n/2)
- Hence *T(n/2) ≤ c (n/2) lg (n/2)*
- Boundary condition can be shown to hold for $n \geq n_0$
- *Difficult for n=1 but holds for n=2 onwards with c ≥ 2*
- Substituting into the recurrence yields
    - *T(n) ≤ 2(c (n/2) lg (n/2)) + n*
      ≤ c n lg (n/2)) + n
      = cn lg n –cn lg 2 + n = cn lg n – cn + n
      ≤ c n lg n (proved by induction)

Substitution method

- *T(n)=2 T(√n) + lg n*
- Renaming m = lg n, $T(2^m) = 2\,T(2^{m/2}) + m$
- Next rename $T(2^m) = S(m)$ to get
  - $S(m) = 2\,S(m/2) + m$
- *S(m) = O(m lg m) alike the earlier example.*
- *Hence T(n)=* $T(2^m) = S(m) = O(m \lg m) = O(\lg n \lg \lg n)$

Iteration method

$T(n) = 3\,T(n/4) + n$ ----- now iterate steps of this

$= n + 3((n/4) + 3\,T(n/16))$

$= n + 3((n/4) + 3(n/16) + 3T(n/64))$

$= n + 3(n/4) + 9(n/16) + 27\,T(n/64)$

So i-th term is $3^i (n/4^i)$

Last term is $3^{(\log_4 n)}\Theta(1) = \Theta(n^{(\log_4 3)})$

Iteration hits $n=1$ when $(n/4^i) = 1$ or $i > \log_4 n$

Hence $T(n)$ is a geometric series of $(3/4)^i$

This yields $T(n) \leq 4n + o(n) = O(n)$

Recursion tree

Convenient way to visualize the recursions

Consider $T(n) = 2\,T(n/2) + n^2$

At the topmost level, excess work is n^2

The next level will need $(n/2)^2 + (n/2)^2 = n^2/2$

The next level will need $4\,(n/4)^2 = n^2/4$

This continues to lg n levels, total work $O(n^2)$

Recursion tree

Consider $T(n) = T(n/3) + T(2n/3) + n$

Here at every level, work amount is n.

So it is important to find the height of this tree

Longest path from root to leaf is $(2/3)^k n = 1$

So the height of the tree is $k = \log_{3/2} n$

Solution of the recurrence is at most $n \log_{3/2} n$

This can be considered as O(n lg n)

Master theorem

Theorem *Let a be an integer greater than or equal to 1 and b be a real number greater than 1. Let c be a positive real number and d a nonnegative real number. Given a recurrence of the form*

$$T(n) = \begin{cases} aT(n/b) + n^c & \text{if } n > 1 \\ d & \text{if } n = 1 \end{cases}$$

then for n a power of b,

1. *if* $\log_b a < c$, $T(n) = \Theta(n^c)$,
2. *if* $\log_b a = c$, $T(n) = \Theta(n^c \log n)$,
3. *if* $\log_b a > c$, $T(n) = \Theta(n^{\log_b a})$.

Outline of proof

Proof: In this proof, we will set $d = 1$, so that the bottom level of the tree is equally well computed by the recursive step as by the base case. It is straightforward to extend the proof for the case when $d \neq 1$.

Let's think about the recursion tree for this recurrence. There will be $\log_b n$ levels. At each level, the number of subproblems will be multiplied by a, and so the number of subproblems at level i will be a^i. Each subproblem at level i is a problem of size (n/b^i). A subproblem of size n/b^i requires $(n/b^i)^c$ additional work and since there are a^i problems on level i, the total number of units of work on level i is

$$a^i(n/b^i)^c = n^c\left(\frac{a^i}{b^{ci}}\right) = n^c\left(\frac{a}{b^c}\right)^i.$$

Terminating condition

$$
\begin{aligned}
& \left(\frac{a}{b^c}\right) = 1 \\
\Leftrightarrow \quad & a = b^c \\
\Leftrightarrow \quad & \log_b a = c \log_b b \\
\Leftrightarrow \quad & \log_b a = c
\end{aligned}
$$

In general, we have that the total work done is

$$\sum_{i=0}^{\log_b n} n^c \left(\frac{a}{b^c}\right)^i = n^c \sum_{i=0}^{\log_b n} \left(\frac{a}{b^c}\right)^i$$

Proof of first two cases

In case 1, $$n^c \sum_{i=0}^{\log_b n} \left(\frac{a}{b^c}\right)^i = \Theta(n^c).$$

In Case 2 we have that $\frac{a}{b^c} = 1$ and so

$$n^c \sum_{i=0}^{\log_b n} \left(\frac{a}{b^c}\right)^i = n^c \sum_{i=0}^{\log_b n} 1^i = n^c(1 + \log_b n) = \Theta(n^c \log n) .$$

Proof of third case

In Case 3, we have that $\frac{a}{b^c} > 1$. So in the series

$$\sum_{i=0}^{\log_b n} n^c \left(\frac{a}{b^c}\right)^i = n^c \sum_{i=0}^{\log_b n} \left(\frac{a}{b^c}\right)^i,$$

the largest term is the last one, the sum is $\Theta\left(n^c \left(\frac{a}{b^c}\right)^{\log_b n}\right)$

$$\begin{aligned} n^c \left(\frac{a}{b^c}\right)^{\log_b n} &= n^c \frac{a^{\log_b n}}{(b^c)^{\log_b n}} \\ &= n^c \frac{n^{\log_b a}}{n^{\log_b b^c}} \\ &= n^c \frac{n^{\log_b a}}{n^c} \\ &= n^{\log_b a}. \end{aligned}$$

Thus the solution is $\Theta(n^{\log_b a})$.

Generalization

Theorem *Let a and b be positive real numbers with $a \geq 1$ and $b \geq 2$. Let $T(n)$ be defined by*

$$T(n) = \begin{cases} aT(\lceil n/b \rceil) + f(n) & \text{if } n > 1 \\ d & \text{if } n = 1. \end{cases}$$

Then

1. *if $f(n) = \Theta(x^c)$ where $\log_b a < c$, then $T(n) = \Theta(n^c) = \Theta(f(n))$.*
2. *if $f(n) = \Theta(n^c)$, where $\log_b a = c$, then $T(n) = \Theta(n^{\log_b a} \log_b n)$*
3. *if $f(n) = \Theta(n^c)$, where $\log_b a > c$, then $T(n) = \Theta(n^{\log_b a})$*

The same results apply with ceilings replaced by floors.

Complexity and randomization

- Running time of PARTITION on subarray of size n will be Θ (n)
- Worst case: T(n) = T(n-1)+Θ (n) = Θ (n^2)
- Best case: T(n) = 2 T(n/2)+ Θ (n) = Θ (n lg n)

To find average case, we need to randomize.

```
RANDOMIZED_PARTITION(A,p,r)
    i=RANDOM(p,r)
    exchange A[p] with A[i]
    return PARTITION(A,p,r)
```

Average case complexity of Randomized Quick Sort

$T(n) = (1/n)\,(T(1)+T(n-1) + \sum^{n-1}_{q=1} T(q) + T(n-q)) + \Theta(n)$

Now, $(1/n)(T(1)+T(n-1))=(1/n)(O(1)+O(n^2))=\Theta(n)$

Hence, due to symmetry wrt q,

$T(n) = (2/n) \sum^{n-1}_{k=1} T(k) + \Theta(n)$

Assume inductively that $T(n) \leq a\, n \lg n + b$

Then $\sum k \lg k \leq \frac{1}{2}\, n^2 \lg n - 1/8\, n^2$ which is bounded

This establishes the $\Omega(n \lg n)$ bound.

Lower bound for sorting algorithms that employ comparisons

Result: Any decision tree that sorts n elements has a height of $h = \Omega(n \lg n)$

Proof:

There can be n! permutations of n elements.

Hence there are n! leaves of the decision tree

For binary tree, $n! \leq 2^h$ or, $h \geq \lg(n!)$

Use Stirling approximation, $n! > (n/e)^n$

To get $h > n \lg n$ which leads to the result.

Counting Sort and Radix Sort

COUNTING-SORT (A,B,k)

```
for i=1 to k
        C[i]=0
for j=1 to length[A]
        C[A[j]]=C[A[j]]+1
for i=2 to k
        C[i]=C[i]+C[i-1]
for j=length[A] downto 1
        B[C[A[J]]]=A[j]
        C[A[j]]=C[A[j]]-1
```

When k-O(n), T(n)=O(n) – linear

Radix-Sort(A,d)

```
for i=1 to d
        use stable counting sort
        on digits from right to left, keeping
        the other digits intact
```

- A= [3^1 6 4^1 1^1 3^2 4^2 1^2 4^3] (array to be sorted)
- C= [2 0 2 3 0 1] (count)
- C= [2 2 4 7 7 8] (cumulative)
- B= [- - - - - - - -] (initial)

- B= [- - - - - - 4^3 -] C= [2 2 4 6 7 8]
- B= [- 1^2 - - - - 4^3 -] C=[1 2 4 6 7 8]
- B= [- 1^2 - - - 4^2 4^3 -] C= [1 2 4 5 7 8]
- B= [- 1^2 - 3^2 - 4^2 4^3 -] C=[1 2 3 5 7 8]
- B= [1^1 1^2 - 3^2 - 4^2 4^3 -] C=[0 2 3 5 7 8]
- B= [1^1 1^2 – 3^2 4^1 4^2 4^3 -] C=[0 2 3 4 7 8]
- B= [1^1 1^2 – 3^2 4^1 4^2 4^3 6] C=[0 2 3 4 7 7]
- B= [1^1 1^2 3^1 3^2 4^1 4^2 4^3 6] C=[0 2 2 4 7 7]

B holds the sorted output

Linear time sorting – Bucket sort

```
BUCKET-SORT (A)
    n = length(A)
    for i= 1 to n
        insert A[i] into list B[floor(n A[i])]
    for i=0 to n-1
        sort list B[i] using insertion sort
Concatenate lists B[0], B[1], .. , B[n-1] together
```

Bucket sort – time complexity

- Here n_i is a random variable denoting |B[i]|, the size of the i^{th} bucket.
- Insertion sort runs in quadratic time, so that expected time to sort is $\sum O(E(n_i^2))$ summed over all n buckets i.e. i= 0 to n-1.
- Now probability that n_i=k follows binomial distribution with p=1/n since there are n elements and n buckets.
- Hence $E[n_i]$= np=1 and $Var[n_i]$= np(1-p)=1-1/n
- So, $E[n_i^2] = 2 – 1/n = \Theta(1)$.
- Hence total expected time for bucket sort is O(n)

Medians and order statistics

```
RANDOMIZED-SELECT(A,p,r,i)
    if p==r return(A[p])
    q=RANDOMIZED-PARTITION(A,p,r)
    k=q-p+1
    if i ≤ k
          return RANDOMIZED-SELECT(A,p,q,i)
    else
          return RANDOMIZED-SELECT(A,q+1,r,i-k)

RANDOMIZED_PARTITION(A,p,r)
    i=RANDOM(p,r)
    exchange A[p] with A[i]
    return PARTITION(A,p,r)
```

Time Complexity of Selection

To find an upper bound T(n) on expected time

$T(n) \leq 1/n\ (T(\max(1,n-1))+\sum T(\max(k,n-k)))+O(n)$

$\leq 1/n\ (T(n-1)+ 2 \sum T(k))+O(n)$

[max(k,n-k) = k or n-k split at k=n/2]

In worst case, $T(n-1) = O(n^2)$ so that $T(n-1)/n = O(n)$

To solve the recurrence- Substituting $T(n) \leq cn$, and noting that the sum over T(k) runs from n/2 to n;

$T(n) \leq 2c/n\ (\frac{1}{2}\ n(n-1) - \frac{1}{2}(n/2)\ ((n/2)-1)) + O(n)$

$T(n) \leq c(3/4\ n - \frac{1}{2}\ )+O(n) \leq cn$ picking c large enough

Worst case linear time selection

1. Divide n elements into n/5 groups of 5 elements each, n mod 5 in last group	• O(n)
2. Find median of each group using insertion sort and taking middlemost element. For even size in last group, take larger of the two medians.	• O(n) calls of O(1) size set insertion sorts
3. Use SELECT recursively to find median x of these n/5 medians	• T(n/5)
4. Partition input array around median of medians x as the pivotal element, no of elements on low side being k	• O(n)
5. Use SELECT recursively to find i-th smallest element on low side if $i \leq k$ or (i-k) th element on high side if $i > k$	• T(7n/10 +6) see next page…

Time complexity – median of medians

- At least half of medians found in Step-2 are ≥ x
- At least half of the groups contribute 3 elements >x, except its own group and the last one i.e. at least 3(½(n/5)-2)
- Hence no of elements > x or < x is at least = 3n/10 -6
- Hence SELECT is called recursively in STEP 5 on at most 7n/10 + 6 elements
- T(n) = O(1) if n ≤ 80 and n>20
- T(n) ≤ T(n/5)+T(7n/10+6)+O(n) if n> 80
- Assume T(n) ≤ cn to show T(n) ≤ 9cn/10 + 7c + O(n)≤ cn

One example case for SELECT

One iteration on a randomized set of 100 elements from 0 to 99

	12	15	11	2	9	5	0	7	3	21	44	40	1	18	20	32	19	35	37	39
	13	16	14	8	10	26	6	33	4	27	49	46	52	25	51	34	43	56	72	79
Medians	17	23	24	28	29	30	31	36	42	47	50	55	58	60	63	65	66	67	81	83
	22	45	38	53	61	41	62	82	54	48	59	57	71	78	64	80	70	76	85	87
	96	95	94	86	89	69	68	97	73	92	74	88	99	84	75	90	77	93	98	91