Homework 1

- 1-1 . Decide whether you think the following statement is true or false. If it is true, give a short explanation. If it is false, give a counterexample.
    - True or false? In every instance of the Stable Matching Problem, there is a stable matching containing a pair(m, w) such that m is ranked first on the preference list of w and w is ranked first on the preference list of m.

  Solution

  False.

  Providing an example,

  consider a group containing m1, m2, w1, w2.

  Preference list :

  m1 = { w1, w2 }

  m2 = { w2, w1 }

  w1 = { m2, m1 }

  w2 = { m1, m2 }

  in this situation, there is no pair(m, w) such that m is ranked first on the preference list of w and w is ranked list on preference list of m.

- 1-2 . Decide whether you think the following statement is true or false. If it is true, give a short explanation. If it is false, give a counterexample.
    - True or false? Consider an instance of the Stable Matching Problem in which there exists a man m and a woman w such that m is ranked first on the preference list of w and w is ranked first on the preference list of m. Then in every stable matching S for this instance, the pair (m, w) belongs to S.

  Solution

  True.

  Consider a perfect match which contains pair (m, w') and pair (m', w), but not containing pair (m, w). Then as all we know that m and w each rank the other first, so they prefer each other rather than their partners (w', m'). As a result , this matching is not stable.

- 1-3 . There are many other settings in which we can ask questions related to some type of "stability" principle. Here's one, involving competition between two enterprises.
    - .....

  Solution

  (b)

  Providing an example,

  Let both networks have two shows each

Network A shows has rating 20 and 40,

Network B shows has rating 30 and 50,

Then we have schedule:

pair 1 { A : 20, B : 30 } (B wins)

pair 2 { A : 40, B : 50 } (B wins)

so B will win all pair, Now A changes its schedule

pair 1 { A : 40, B : 30 } (A wins)

pair 2 { A : 20, B : 50 } (B wins)

Now both win one time, and B changes to win all pair

as a result, there is no stable matching.

- 2-1. Suppose you have algorithms with the five running times listed below.(Assume these are the exact running times.) How much slower do each of these algorithms get when you (a) double the input size, or (b) increase the input size by one?

  (a) n^2

  (b) n^3

  (c) $100\ n^2$

  (d) $nlogn$

  (e) 2^n

  Solution

  (a)

  (a) n^2 —>$(2n)^2$ —> $4n^2$ it takes 4 times more time

  (b) n^3 —>$(2n)^3$ —>$8n^3$ it takes 8 times more time

  (c) $100n^2$ —>$100(2n)^2$ —>$400n^2$ it takes 4 times more time

  (d) $nlogn$ —>$2nlog2n$ —>$2n(logn+2)$ it takes 2 times more time

  (e) 2^n —>2^{2n} it takes square of the previous running time.

  (b)

  (a) an additive 2n+1

  (b) an additive $3n^2+3n+1$

  (c) an additive 200n + 100

  (d) an additive $log(n+1)+n[log(n+1)-logn]$

  (e) it takes twice more

- 2-2.Suppose you have algorithms with the six running times listed below. (Assume these are the exact number of operations performed as a function of the input size n.) Suppose you have a computer that can perform 10^{10} operations per second, and you need to compute a result in at most an hour of computation. For each of the algorithms, what is the largest input size n for which you would be able to get the result within an hour?

(a) n^2
(b) n^3
(c) $100n^2$
(d) $nlogn$
(e) 2^n
(f) 2^{2^n}

Solution

(a) 6×10^6

(b) 33019

(c) 6×10^4

(d) $\approx 9 \times 10^{11}$

(e) 45

(f) 5

- 2-3. Take the following list of functions and arrange them in ascending order of growth rate. That is, if function g(n) immediately follows function f(n) in your list, then it should be the case that f(n) is O(g(n)).

  $f_1(n) = n^{2.5}$

  $f_2(n) = \sqrt{2n}$

  $f_3(n) = n + 10$

  $f_4(n) = 10^n$

  $f_5(n) = 100^n$

  $f_6(n) = n^2 logn$

  Solution

  $f_2 < f_3 < f_6 < f_1 < f_4 < f_5$

  ------polynomials-------

  $f_2 < f_3 : \ n^{\frac{1}{2}} < n$

  $f_3 < f_6 : \ n < n^2 logn$

  $f_6 < f_1 : \ n^2 \times n^{\frac{1}{2}} < n^2 \times logn$

  ------exponentials--------

  $f_1 < f_4 : \ n^{2.5} < 10^n$

  $f_4 < f_5 : \ 10^n < 100^n$

- 2-4. Take the following list of functions and arrange them in ascending order of growth rate. That is, if function g(n) immediately follows function f(n) in your list, then it should be the case that f(n) is O(g(n)).

  $g_1(n) = 2^{\sqrt{logn}}$

  $g_2(n) = 2^n$

  $g_3(n) = n(logn)^3$

  $g_4(n) = n^{\frac{4}{3}}$

  $g_5(n) = n^{logn}$

$g_6(n) = 2^{2^n}$

$g_7(n) = 2^{n^2}$

Solution

$g_1 < g_5 < g_3 < g_4 < g_2 < g_7 < g_6$

- 2-5. Assume you have functions f and g such that f(n) is O(g(n)). For each of the following statements, decide whether you think it is true or false and give a proof or counterexample.

  (a) $log_2 f(n)\ is\ O(log_2 g(n))$

  (b) $2^{f(n)}\ is\ O(2^{g(n)})$

  (c) $f(n)^2\ is\ O(g(n)^2)$

  Solution

  (a) False, When g(n) = 1, then $log_2 g(n) = O$, so $log_2 f(n) \nleq clog_2 g(n)$

  (b) False, If f(n) = 2n, g(n) = n , then $2^{f(n)} = 4^n$, $2^{g(n)} = 2^n$

  (c) True because $f(n) \leq cg(n) \quad \forall\, n \geq\ n_0$

  so $(f(n))^2 \leq c^2(g(n))^2 \quad \forall\, n \geq\ n_0$

- 2-6. Consider the following basic problem. You're given an array A consisting of n integers A[1], A[2], ... , A[n]. You'd like to output a two-dimensional n-by-n array B in which B[i, j] (for i < j) contains the sum of array entries A[i] through A[j]—that is, the sum A[i] + A[i + 1] + ...+ A[j]. (The value of array entry B[i, j] is left unspecified whenever i ≥ j, so it doesn't matter what is output for these values.)

  Here's a simple algorithm to solve this problem.

  For i = 1, 2, ..., n

  For j = i + 1, i + 2, . . . , n

  Add up array entries A[i] through A[j]

  Store the result in B[i,j]

  Endfor

  Endfor

  (a) For some function f that you should choose, give a bound of the form O(f (n)) on the running time of this algorithm on an input of size n (i.e., a bound on the number of operations performed by the algorithm).
  (b) For this same function f , show that the running time of the algorithm on an input of size n is also □(f (n)). (This shows an asymptotically tight bound of □(f (n)) on the running time.)
  (c) Although the algorithm you analyzed in parts (a) and (b) is the most natural way to solve the problem—after all, it just iterates through the relevant entries of the array B, filling in a value for each—it contains some highly unnecessary sources of inefficiency. Give a different algorithm to solve this problem, with an asymptotically better running time. In other words, you should design an algorithm with running time O(g(n)), where limn→∞ g(n)/f (n) = 0.

Solution

a) Outer Loop runs exactly **n** iterations

Inner Loop runs at most **n** iterations

Storing result in B[i,j] requires constant time

so, Running time = $n^2 O(n) \equiv O(n^3)$

b) consider the case when $i \leq \frac{n}{4} \ \ and \ \ j \geq \frac{3n}{4}$

$j - i + 1 \geq \frac{3n}{4} - \frac{n}{4} + 1 > \frac{n}{2}$

Adding array entries [A[i]] through [A[j]] would require $\frac{n}{2}$ operation.

There are $\frac{n}{4}^2$ such pairs.

so, the algorithm must perform $\frac{n}{2} \cdot \frac{n}{4}^2 \ = \ \frac{n^3}{32}$ operations.

This is $\Omega(n^3)$

(c) for i = 1, 2, 3, ...n

B[i, i+1] $\leftarrow$ A[i] + A[i+1]

for k = 2, 3, ..., n-1

for i = 1, 2, 3,....., n-k

j = i + k

B[i, j] $\leftarrow$ B[i, j-1] + A[j]