**CMPE 300 ANALYSIS OF ALGORITHMS**

###### FINAL ANSWERS

function SortCRCW (L[1:n])

Model: CRCW PRAM with p=n(n-1)/2 processors, write conflicts are resolved by summing

Input: L[1:n] (an array of n elements)

Output: L[1:n] (sorted list in descending order)

 for 1≤i≤n do in parallel

 Win [i] = 0

 end in parallel

 for 1≤i,j≤n and i<j do in parallel

 if (L[i] > L[j]) then

 Win [j] = 1

 else

 Win [i] = 1

 end if

 end in parallel

 for 1≤i≤n do in parallel

 L [Win [i] + 1] = L[i]

 end in parallel

end

The concurrent write strategy is “summing the write operations”, which is a combining CW strategy. Any element L[i] is compared with all other elements and after the comparisons Win[i] holds the number of elements from which L[i] is smaller. Thus, Win[i]+1 gives the correct position of L[i] in the output.

Basic operation: Parallel comparison statement

In the second parallel loop, there is just 1 parallel comparison. So, W(n) = 1 ϵ Ө(1).

S(n) = W\*(n) / W(n) = (n\*log n) / 1 = n\*log n ϵ Ө(n\*log n)

C(n) = p(n) \* W(n) = n(n-1)/2 \* 1 = n(n-1)/2 ϵ Ө(n2)

E(n) = S(n) / p(n) = (n\*log n) / (n(n-1)/2) = (2\*log n)/(n-1)

The algorithm is not cost optimal, since C(n) = n(n-1)/2 > W\*(n) = n\*log n.

The algorithm is not cost efficient, since C(n) is not ϵ O( (n\*log n) \* log k (n\*log n)).

The algorithm is time efficient, since W(n) = 1 ϵ O(log k n) for some k≥0.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| i | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| j=0 | x=8 | x=7 | x=6 | x=5 | x=4 | x=3 | x=2 | x=1 |
|  | y=7 | y=6 | y=5 | y=4 | y=3 | y=2 | y=1 | y=0 |
|  | M[2]=15 | M[3]=13 | M[4]=11 | M[5]=9 | M[1]=1 | M[1]=1 | M[1]=1 | M[1]=1 |
|  | So, array M will be : [1 – 15 – 13 – 11 – 9 – 3 – 2 – 1 – 0 ...] |
| j=1 | x=1 | x=15 | x=13 | x=11 | x=9 | x=3 | x=2 | x=1 |
|  | y=13 | y=11 | y=9 | y=3 | y=2 | y=1 | y=0 | y=0 |
|  | M[3]=14 | M[4]=26 | M[5]=22 | M[6]=14 | M[2]=7 | M[2]=2 | M[2]=2 | M[2]=1 |
|  | So, array M will be : [1 – 7 – 14 – 26 – 22 – 14 – 2 – 1 – 0 ...] |
| j=2 | x=1 | x=7 | x=14 | x=26 | x=22 | x=14 | x=2 | x=1 |
|  | y=22 | y=14 | y=2 | y=1 | y=0 | y=0 | y=0 | y=0 |
|  | M[5]=23 | M[6]=21 | M[7]=16 | M[8]=27 | M[4]=22 | M[4]=14 | M[4]=2 | M[4]=1 |
|  | So, array M will be : [1 – 7 – 14 – 22 – 23 – 21 – 16 – 27 – 0 ...] |

So, the answer is:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Position | M[1] | M[2] | M[3] | M[4] | M[5] | M[6] | M[7] | M[8] | M[9] | M[10] | M[11] |
| Value | 1 | 7 | 14 | 22 | 23 | 21 | 16 | 27 | 0 | 0 | ... |

1. During the execution of the algorithm (any algorithm), the adversary employs the following rule: If currently, according to the previous questions and answers, the number should be between a and b, then when the algorithm asks “is the number c?” (a≤c≤b), the adversary changes the selected number such that it will be a number between a and c if |a-c|>|c-b| and between c and b otherwise. In this way, each question can at most eliminate half of the numbers in the range where the selected number should be. We can show easily by a recurrence relation that thus the number of steps (questions) required must be at least log n. So, the lower bound of this problem is Ω(log n).
2. function Compare (A[1:n], B[1:n])

 call random ({1,...,n}, i)

 if A[i]=B[i] then

 return true

 else

 return false

 endif

end

When the algorithm returns false, we are sure that the arrays are not equal and thus the answer is correct. However, when it returns true, the arrays may or may not be equal. So, the algorithm is false-biased.

Since the algorithm includes randomness and may return an incorrect output, it is a probabilistic Monte Carlo algorithm.

Probability that the output is incorrect when the algorithm returns true: In the worst scenario, n-1 pairs will be equal but the arrays will not be equal. The probability of returning incorrect output (i.e. returning true) in this case is (n-1)/n. That is, (since this is the worst scenario) the probability of making an error is at most (n-1)/n. Therefore, the probability of returning correct answer is at least 1-(n-1)/n = 1/n. That is, the algorithm is 1/n-correct.

Basic operation is comparison. Clearly, W(n)=1 ϵ Ө(1).

1. function Compare (A[1:n], B[1:n])

 while (true) do

 call random ({1,...,n}, i)

 if A[i]≠B[i] then

 return false

 endif

 end while

end

When the algorithm returns an output (which is the output false), we are sure that the arrays are not equal. However, there is a chance that the algorithm does not terminate. Thus, the algorithm either gives a correct answer or it does not terminate. So, it is a Las Vegas algorithm.

For an input I, suppose that k out of n elements in the arrays are not equal (k≤n). Then

probability that algorithm will terminate in a pass: p=k/n

probability that i passes will be performed: pi=(1-p)i-1\*p

$$T\_{exp}(I)=\sum\_{i=1}^{\infty }i\*p\_{i}=\sum\_{i=1}^{\infty }i\*\left(1-p\right)^{i-1}\*p=p\left(\frac{1}{1-\left(1-p\right)}\right)^{2}=\frac{1}{p}=\frac{n}{k}$$

(For worst case, k=0. For best case, k=n.)