Algorithms Tutorial 3 (Worksheet)

Page No.
Date :

Problem: To find a maximum sum subarray in the given array.
Input : Array arr[] and size n.

Define function max_stock (int arr[], int l, int r)
{ define int max_val;
Calculate mid as $l+(r-l)/2$; (l: lower bound, r: upper bound.)
If Base case (One element) (ie l == r):
{ max_val = arr[l] }
else, recursively call this function as follows:
{ left_stock_max = max_stock (arr, l, mid)
right_stock_max = max_stock (arr, mid+1, r)

// we also need a max value subarray that
// contains elements both, below index mid and above
// index mid too.
left_right_stock_max = max_left_right (arr, l, m, r)
max_val = maximum among (left_stock_max,
calculates maximum of 3 elements. right_stock_max,
left_right_stock_max)
}
return max_val;
}

Define max_left_right (int arr[], int l, int m, int r)
{
define int sum=0, left_sum=0, right_sum = 0;
loop for elements from index m to l with loop variable
i and each step i--
{
Add arr[i] to sum;
If sum > left_sum then assign:
left_sum = sum.
}

make sum=0 again;
loop for elements from index mid+1 to r with loop variable i and i++ at each step
{
    Add arr[i] to sum;
    If sum > right_sum, then assign:
        right_sum = sum
}
Return maximum among ~~(left_sum~~ (left_sum, right_sum, (left_sum + right_sum))
↓ calculates maximum among 3 elements.

void main ()
{
~~Call function max_stock~~
Input n and array arr;
Call function max_stock (arr, 0, n-1) and print value returned by it
}

ALGORITHM OVER

calculating subarray across mid, that has elements both below ~~mid~~ index mid and above index mid in it.

Time complexity ⇒ $T(n) = 2T(n/2) + \Theta(n)$

↓ calculating left subarray and right subarray.

This is similar to merge sort and Hence we get

~~Best $\Theta(n \log n)$ Worst~~

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