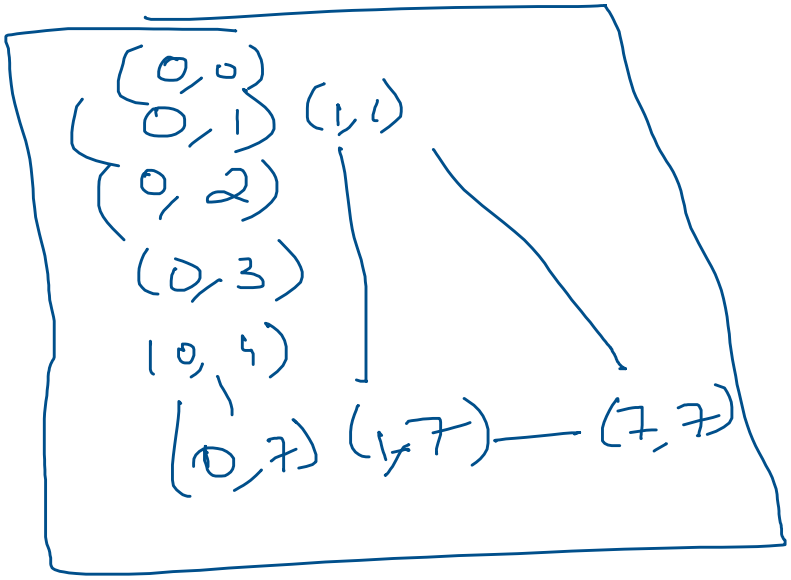
June 30 D-Day!

30 June 2020 21:13

Max Sum Subarray

i	j						
0	1	2	3	4	5	6	7
-2	-3	4	-1	-2	1	5	-3



Brute Force

```
for(i)
{for(j)
 {
    for(sum)
      {Max Store;
 }    }
```

$O(n^3)$

}

Optimization - 1 → $O(n^2)$

```
for(i)
  for(j)
  {
      sum = sum + arr[j];
      Every time store the max sum;
  }
```

Optimization - 2

-5 1 -3 4

Max = -2

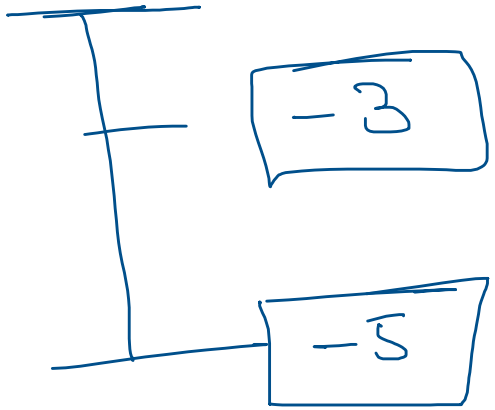
0	1	2	3	4	5	6	7
-2	-3	4	-1	-2	1	5	-3

D.P

Extra Space →

-2	-3	4	3	1	2	7	4

← Max → MaxSoFar



curr_MAX (Ans)

Be a part of already existing subarray

Start a new subarray from yourself

Optimization -3

This optimization handles all the scenarios even if all elements are negative.

0	1	2	3	4	5	6	7
-2	-3	4	-1	-2	1	5	-3

4

-2 + -3 → -5

-3

-3

MAX_SO_FAR = arr[0]; 7

curr_max = arr[0];

```
for (int i=0 ; i<size ; i++)
{
    curr_max = Math.max(arr[i], arr[i] + curr_max);
    max_so_far = Math.max(curr_max, max_so_far);
}
```

1

Pick the Bar

update

2

return max_so_far;

Kadane's Algo

Does not work when all the elements are negative.
Or We can say that it assumes at least 1 positive element will be there in the array.

0	1	2	3	4	5	6	7
-2	-3	4	-1	-2	1	5	-3
		4	3	1	2	7	4

$\boxed{max_so_far = 7}$

curr_max = 0

curr_max = 4

3

Array Rotation

Rotate array of size $n = 6$ by $d = 2$ elements

space $O(d)$ ← Array

1	2	3	4	5	6

i ↓ end $0 \to i-1$

Same

3	4	5	6	1	2

Time → $O(n)$

Method 1 → Create new array & copy accordingly

Method 2 →

2	3	4	5	6	1

→ Rotate one by 1

$O(1)$

time → $O(n * d)$

Method 3 → Juggling Algorithm

0	1	2	3	4	5
3	4	5	6	1	2

$i = j$

$arr[j] = arr[j+d]$

$d = 2$ rotations

$(j+d)/size = i$

temp = 2

$d, n \rightarrow$ GCD (HCF)

for (i)

{

while (j) →

{

2, 6

d n

2, 7

}

2

5+2=7

set

0

1

3 | 4 | 5 | 6 | 7 | 1 | 2

1

temp = arr[j]

j = 6

2, 7

GCD → 1

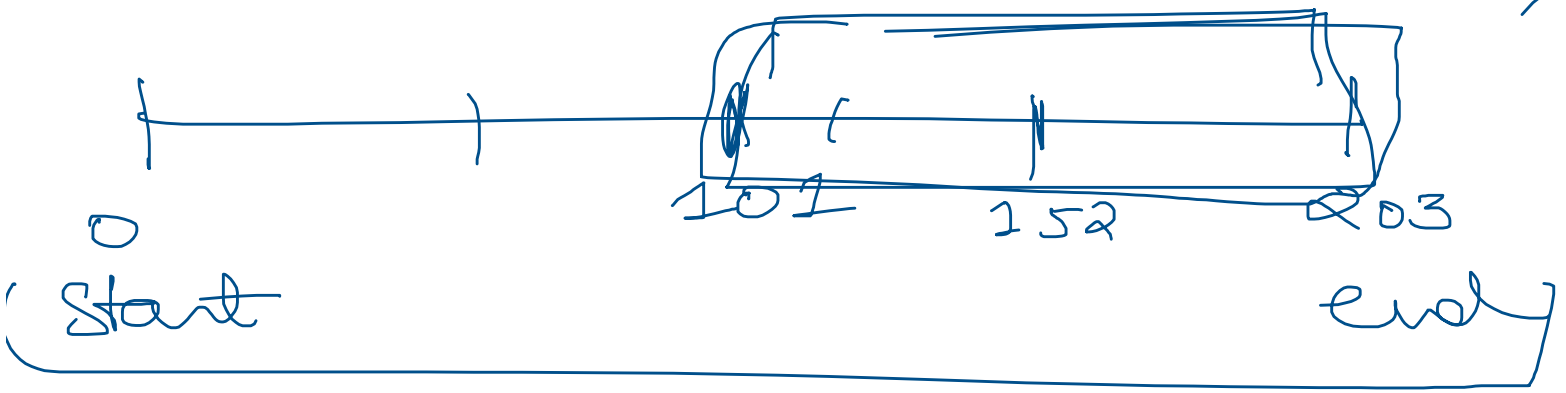
Allocate no. of Pages

101 →

100 Pages 2 Book

↳ 50 — 50

[12, 34, 67, 90] → Total = 203



113

152

True

Total = sum of array

Start = 0 , end = Total

initial conditions

no. of pages = Integer. max-value;

```
while(start <= end)
{   mid = (start + end)/2 ;
    if (isPossibleToRead[arr, n, m, mid])
    {
```

no. of pages = min(no. of pages, mid);

end = mid - 1;

}

else

{

start = mid + 1;

}

}

is Possible to Read(arr, n, m, curr-min)

{ No. of student = 1

curr-sum = 0

Arr[12, 34, 67, 90]

curr

2 students

3 students required

```
if(arr[i] > curr_min)
{return false;
}
if( curr_sum + arr[i] > curr_min)
{
    No. of student ++;
    curr_sum = arr[i];
    if( No. of student > m)
    { return false;
    }
}
else
{ curr_sum = curr_sum + arr[i];
}
```

2nd → 12, 34

67

4