

PAL: Tutorial, Sheet 1

Question 1

$$a(0) = 0 \quad a(1) = 1$$

$$a(n) = a(n-1) + a(n-2) + 1$$

Part 1a - Recursive Sequential Algorithm

```
1  fun calcA(n) {  
2      if n == 0 then return 0  
3      if n == 1 then return 1  
4      return calcA(n - 1) + calcA(n - 2) + 1  
5  }
```

Part 1b - Recursive Parallel Algorithm

```
1  fun calcA(n) {  
2      if n == 0 then return 0  
3      if n == 1 then return 1  
4      x = spawn calcA(n - 1)  
5      y = calcA(n - 2)  
6      sync  
7      return x + y + 1  
8  }
```

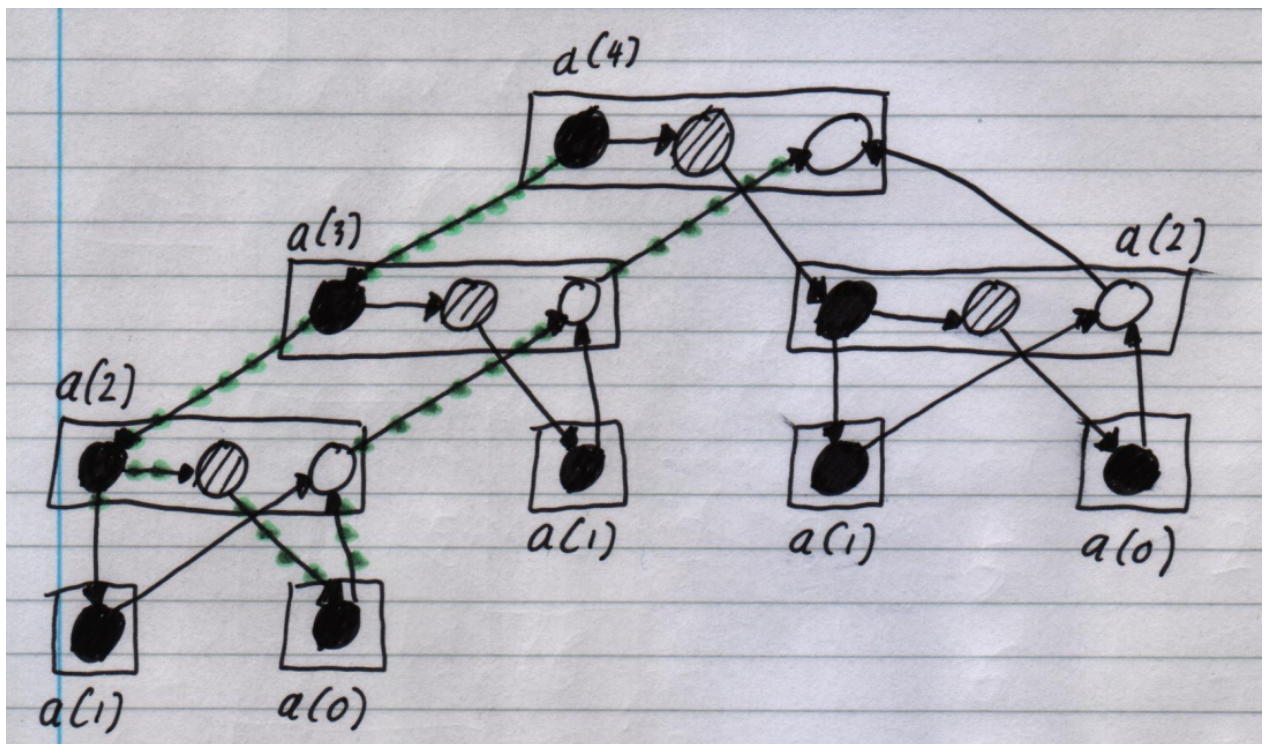
Part 2 - Values

$$\begin{aligned} a(2) &= a(1) + a(0) + 1 \\ &= 1 + 0 + 1 \\ &= 2 \end{aligned}$$

$$\begin{aligned} a(3) &= a(2) + a(1) + 1 \\ &= 2 + 1 + 1 \\ &= 4 \end{aligned}$$

$$\begin{aligned} a(4) &= a(3) + a(2) + 1 \\ &= 4 + 1 + 1 \\ &= 7 \end{aligned}$$

Part 3 - Computation DAG of $a(4)$



Work: 9 calls to `calcA`.

Span: 7 (shown in green).

Question 2

$$a(0) = 0 \quad b(0) = 1$$

$$a(n) = a(n-1) + b(n-1) + 1$$

$$b(n) = b(n-1) + 2a(n-1) + 1$$

Part 1a - Recursive Sequential Algorithm

```

1  fun calcA(n) {
2      if n == 0 then return 0
3      return calcA(n - 1) + calcB(n - 1) + 1
4  }
5
6  fun calcB(n) {
7      if n == 0 then return 0
8      return calcB(n - 1) + (2 * calcA(n - 1)) + 1
9  }

```

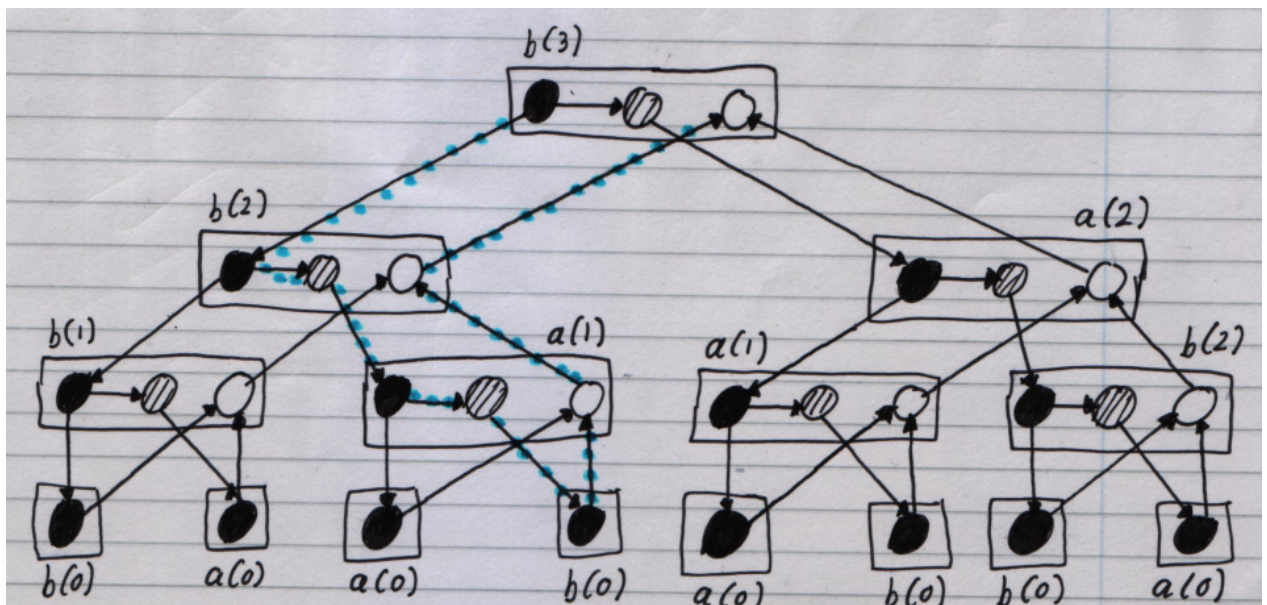
Part 1b - Recursive Parallel Algorithm

```

1  fun calcA(n) {
2      if n == 0 then return 0
3      x = spawn calcA(n - 1)
4      y = calcB(n - 1)
5      sync
6      return x + y + 1
7  }
8
9  fun calcB(n) {
10     if n == 0 then return 0
11     x = spawn calcB(n - 1)
12     y = calcA(n - 1)
13     sync
14     return x + (2 * y) + 1
15 }

```

Part 2 - Computation DAG of $b(3)$



Work: 15 calls to `calcA` or `calcB`.

Span: 8 (shown in blue).

Question 3

Part 1 - Recursive MPSum() for $n = 2^k$

```

1  fun MPSum(array A) {
2      return MPSum(A, 1, A.length)
3  }
4
5  fun MPSum(array A, int s, int e) {
6      if s == e then return A[s]
7
8      n = e - s + 1
9      x = spawn MPSum(A, s, s + n/2 - 1)
10     y = MPSum(A, s + n/2, e)
11     sync
12     return x + y
13 }

```

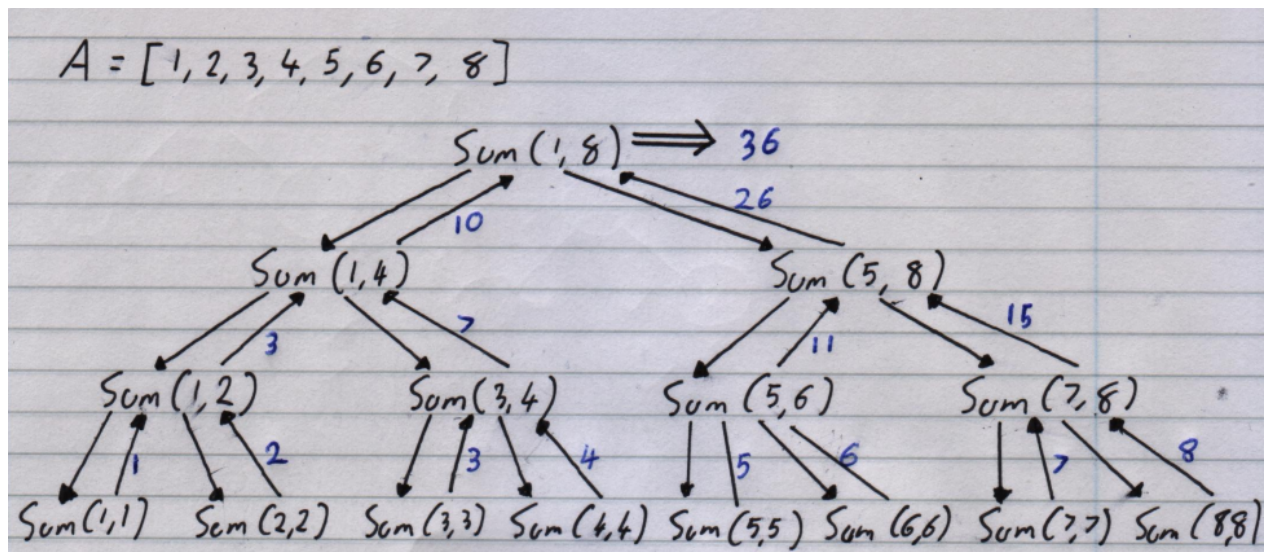
Part 2 - Recursive MPSum() for General n

```

1  fun MPSum(array A) {
2      nextPowerOfTwo = 2 ^ roundUp(log_2(A.length))
3      return MPSum(A, 1, nextPowerOfTwo)
4  }
5
6  fun MPSum(array A, int s, int e) {
7      if s > A.length then return 0
8      if s == e then return A[s]
9
10     n = e - s + 1
11     x = spawn MPSum(A, s, s + n/2 - 1)
12     y = MPSum(A, s + n/2, e)
13     sync
14     return x + y
15 }

```

This algorithm pads with zeros up to the next power of two. The $O(\log_2(n))$ performance is not affected by this approach.

Part 3 - Execution for $A = [1, 2, 3, \dots, 8]$ 

Work for $A[1..n]$ is $O(2^{k+1})$ if $n = 2^k$.

Span for $A[1..n]$ is proportional to $\log_2(n)$.

Part 4 - Recursive `MPMax()` for $n = 2^k$

```

1  fun MPMaX(array A) {
2      return MPMaX(A, 1, A.length)
3  }
4
5  fun MPMaX(array A, int s, int e) {
6      if s == e then return A[s]
7
8      n = e - s + 1
9      x = spawn MPMaX(A, s, s + n/2 - 1)
10     y = MPMaX(A, s + n/2, e)
11     sync
12     if x > y
13         return x
14     else
15         return y
16 }

```

Part 5 - Recursive MPMember() for $n = 2^k$

```
1  fun MPMember(array A, int x) {
2      return MPMember(A, x, 1, A.length)
3  }
4
5  fun MPMember(array A, int x, int s, int e) {
6      if s == e
7          if A[s] == x
8              return true
9          else
10             return false
11
12     n = e - s + 1
13     x = spawn MPMember(A, s, s + n/2 - 1)
14     y = MPMember(A, s + n/2, e)
15     sync
16     return x OR y
17 }
```