Contents

Section outline

1. **More applications of DC**
   - Divide and Conquer Strategy
   - Polynomial multiplication
   - Polynomial multiplication by DC
   - Polynomial multiplication by Karatsuba DC
   - Maximum Sum Subarray by brute force
   - Maximum Sum Subarray by DC
   - Matrix multiplication by DC
   - Practice problems

Divide and Conquer Strategy

1. Given a problem, identify a small number of smaller subproblems of the same type and of similarly sizes
2. Solve each subproblem recursively (the smallest possible size of a subproblem is a base-case)
3. Combine these solutions into a solution for the main problem

$$T(n) = \begin{cases} T_{\text{Divide}} + \sum_{P_i} T(|P_i|) + T_{\text{Combine}} \\ T_{\text{Base}} \end{cases}$$

Polynomial multiplication

- $C(x) = A(x)B(x) = \sum_{i=0}^{2n-2} c_i x_i,\ c_i = \sum_{0 \le j, i-j \le n-1} a_j b_{i-j}$
  [Note the convolution of the coefficients]
- Time complexity of this scheme is:

Polynomial multiplication

- $C(x) = A(x)B(x) = \sum_{i=0}^{2n-2} c_i x_i,\ c_i = \sum_{0 \leq j, i-j \leq n-1} a_j b_{i-j}$
  [Note the convolution of the coefficients]
- Time complexity of this scheme is: $O(n^2)$, actually $\Theta(n^2)$

```
float polyMul(float *A, *B, *C, int deg) {
  int j, k;
  for (j=0; j<=2*deg-2; j++) C[j] = 0;
  for (j=0; j<=deg-1; j++)
    for (k=0; k<=deg-1; k++)
      C[j+k] += A[j] * B[k];
}
```

Polynomial multiplication

- $C(x) = A(x)B(x) = \sum_{i=0}^{2n-2} c_i x_i,\ c_i = \sum_{0 \le j, i-j \le n-1} a_j b_{i-j}$
  [Note the convolution of the coefficients]
- Time complexity of this scheme is: $O(n^2)$, actually $\Theta(n^2)$

```
float polyMul(float *A, *B, *C, int deg) {
  int j, k;
  for (j=0; j<=2*deg-2; j++) C[j] = 0;
  for (j=0; j<=deg-1; j++)
    for (k=0; k<=deg-1; k++)
      C[j+k] += A[j] * B[k];
}
```

- Can we do better than $\Theta(n^2)$, how could we do that?

Polynomial multiplication

- $C(x) = A(x)B(x) = \sum_{i=0}^{2n-2} c_i x_i,\ c_i = \sum_{0 \le j, i-j \le n-1} a_j b_{i-j}$
  [Note the convolution of the coefficients]
- Time complexity of this scheme is: $O(n^2)$, actually $\Theta(n^2)$

```
float polyMul(float *A, *B, *C, int deg) {
  int j, k;
  for (j=0; j<=2*deg-2; j++) C[j] = 0;
  for (j=0; j<=deg-1; j++)
    for (k=0; k<=deg-1; k++)
      C[j+k] += A[j] * B[k];
}
```

- Can we do better than $\Theta(n^2)$, how could we do that?
- Since the time grows with the degree, can we compute the product multiplying smaller polynomials?

Polynomial multiplication by DC

- $A(x) = A_L(x) + x^t A_H(x)$
  $A_L(x) = a_0 + a_1 x + \ldots + a_{t-1} x^{t-1}$,
  $A_H(x) = a_t + a_{t+1} x + \ldots + a_{n-1} x^{n-1-t}$
- $t = \left\lfloor \frac{n}{2} \right\rfloor$, so that both $A_L(x)$ and $A_H(x)$ are nearly equal
- Ideally, n is a power of 2, $n = 2^d, d \geq 0$
- Similarly, $B(x) = B_L + x^t B_H$, where $B_L \equiv B_L(x)$ and $B_H \equiv B_H(x)$
- $C(x) = A(x)B(x) = x^{2t} A_H B_H + x^t (A_H B_L + A_L B_H) + A_L B_L$

Polynomial multiplication by DC

- $A(x) = A_L(x) + x^t A_H(x)$
  $A_L(x) = a_0 + a_1 x + \ldots + a_{t-1} x^{t-1}$,
  $A_H(x) = a_t + a_{t+1} x + \ldots + a_{n-1} x^{n-1-t}$
- $t = \left\lfloor \frac{n}{2} \right\rfloor$, so that both $A_L(x)$ and $A_H(x)$ are nearly equal
- Ideally, n is a power of 2, $n = 2^d, d \geq 0$
- Similarly, $B(x) = B_L + x^t B_H$, where $B_L \equiv B_L(x)$ and $B_H \equiv B_H(x)$
- $C(x) = A(x)B(x) = x^{2t} A_H B_H + x^t (A_H B_L + A_L B_H) + A_L B_L$
- Smaller polynomials recursively multiplied until the degree reduces to 0, when the coefficients are directly multiplied
- $T_{\text{mulHL}}(n) = \begin{cases} a & [n = 1] \\ 4T_{\text{mulHL}}\left(\frac{n}{2}\right) + bn + c & [n > 1, n = 2^d, d \geq 0] \end{cases}$

Polynomial multiplication by DC

- $A(x) = A_L(x) + x^t A_H(x)$
  $A_L(x) = a_0 + a_1 x + \ldots + a_{t-1} x^{t-1}$,
  $A_H(x) = a_t + a_{t+1} x + \ldots + a_{n-1} x^{n-1-t}$
- $t = \left\lfloor \frac{n}{2} \right\rfloor$, so that both $A_L(x)$ and $A_H(x)$ are nearly equal
- Ideally, n is a power of 2, $n = 2^d, d \geq 0$
- Similarly, $B(x) = B_L + x^t B_H$, where $B_L \equiv B_L(x)$ and $B_H \equiv B_H(x)$
- $C(x) = A(x)B(x) = x^{2t} A_H B_H + x^t (A_H B_L + A_L B_H) + A_L B_L$
- Smaller polynomials recursively multiplied until the degree reduces to 0, when the coefficients are directly multiplied
- $T_{\text{mulHL}}(n) = \begin{cases} a & [n = 1] \\ 4T_{\text{mulHL}}\left(\frac{n}{2}\right) + bn + c & [n > 1, n = 2^d, d \geq 0] \end{cases}$
- Solution (by standard methods): $T_{\text{mulHL}}(n) \in \Theta\left(n^2\right)$

Polynomial multiplication by DC

- $A(x) = A_L(x) + x^t A_H(x)$
  $A_L(x) = a_0 + a_1 x + \ldots + a_{t-1} x^{t-1}$,
  $A_H(x) = a_t + a_{t+1} x + \ldots + a_{n-1} x^{n-1-t}$
- $t = \left\lfloor \frac{n}{2} \right\rfloor$, so that both $A_L(x)$ and $A_H(x)$ are nearly equal
- Ideally, n is a power of 2, $n = 2^d, d \geq 0$
- Similarly, $B(x) = B_L + x^t B_H$, where $B_L \equiv B_L(x)$ and $B_H \equiv B_H(x)$
- $C(x) = A(x)B(x) = x^{2t} A_H B_H + x^t (A_H B_L + A_L B_H) + A_L B_L$
- Smaller polynomials recursively multiplied until the degree reduces to 0, when the coefficients are directly multiplied
- $T_{\text{mulHL}}(n) = \begin{cases} a & [n = 1] \\ 4T_{\text{mulHL}}\left(\frac{n}{2}\right) + bn + c & [n > 1, n = 2^d, d \geq 0] \end{cases}$
- Solution (by standard methods): $T_{\text{mulHL}}(n) \in \Theta\left(n^2\right)$
- No improvement, but why?

Polynomial multiplication by DC

- $A(x) = A_L(x) + x^t A_H(x)$
  $A_L(x) = a_0 + a_1 x + \ldots + a_{t-1} x^{t-1}$,
  $A_H(x) = a_t + a_{t+1} x + \ldots + a_{n-1} x^{n-1-t}$
- $t = \left\lfloor \frac{n}{2} \right\rfloor$, so that both $A_L(x)$ and $A_H(x)$ are nearly equal
- Ideally, n is a power of 2, $n = 2^d, d \geq 0$
- Similarly, $B(x) = B_L + x^t B_H$, where $B_L \equiv B_L(x)$ and $B_H \equiv B_H(x)$
- $C(x) = A(x)B(x) = x^{2t} A_H B_H + x^t (A_H B_L + A_L B_H) + A_L B_L$
- Smaller polynomials recursively multiplied until the degree reduces to 0, when the coefficients are directly multiplied
- $T_{\text{mulHL}}(n) = \begin{cases} a & [n = 1] \\ 4T_{\text{mulHL}}\left(\frac{n}{2}\right) + bn + c & [n > 1, n = 2^d, d \geq 0] \end{cases}$
- Solution (by standard methods): $T_{\text{mulHL}}(n) \in \Theta\left(n^2\right)$
- No improvement, but why? There are too many sub-problems

Polynomial multiplication by Karatsuba DC

- $C(x) = A(x)B(x) = x^{2t}A_HB_H + x^t(A_HB_L + A_LB_H) + A_LB_L$

Polynomial multiplication by Karatsuba DC

- $C(x) = A(x)B(x) = x^{2t}A_HB_H + x^t\,(A_HB_L + A_LB_H) + A_LB_L$
- $A_HB_L + A_LB_H = (A_H + A_L)(B_H + B_L) - A_HB_H - A_LB_L$
- Store and reuse A_HB_H and A_LB_L computed earlier
- Smaller polynomials recursively multiplied until the degree reduces to 0, when the coefficients are directly multiplied

Polynomial multiplication by Karatsuba DC

- $C(x) = A(x)B(x) = x^{2t} A_H B_H + x^t (A_H B_L + A_L B_H) + A_L B_L$
- $A_H B_L + A_L B_H = (A_H + A_L)(B_H + B_L) - A_H B_H - A_L B_L$
- Store and reuse $A_H B_H$ and $A_L B_L$ computed earlier
- Smaller polynomials recursively multiplied until the degree reduces to 0, when the coefficients are directly multiplied
- $T_{\text{mulHL}}(n) = \begin{cases} a & [n = 1] \\ 3\,T_{\text{mulHL}}\left(\frac{n}{2}\right) + b'n + c' & [n > 1, n = 2^d, d \geq 0] \end{cases}$
- More polynomial additions, so the constants b' and c' are expected to be somewhat larger than before

Polynomial multiplication by Karatsuba DC

- $C(x) = A(x)B(x) = x^{2t}A_HB_H + x^t(A_HB_L + A_LB_H) + A_LB_L$
- $A_HB_L + A_LB_H = (A_H + A_L)(B_H + B_L) - A_HB_H - A_LB_L$
- Store and reuse A_HB_H and A_LB_L computed earlier
- Smaller polynomials recursively multiplied until the degree reduces to 0, when the coefficients are directly multiplied
- $T_{\text{mulHL}}(n) = \begin{cases} a & [n=1] \\ 3T_{\text{mulHL}}\left(\frac{n}{2}\right) + b'n + c' & [n>1, n=2^d, d \geq 0] \end{cases}$
- More polynomial additions, so the constants b' and c' are expected to be somewhat larger than before
- Solution (by standard methods): $T_{\text{mulHL}}(n) \in \Theta\left(n^{\log_2 3}\right)$, $\log_2 3 = 1.58496\ldots$

Maximum Sum Subarray by brute force

- You are given a one dimensional array that may contain both positive and negative integers
- Find the sum of contiguous subarray of numbers which has the largest sum, e.g.: {-3, -6, **7, -1, -3, 2, 5**, -7}
- Time taken is for finding sum of intervals and computing their max
- $T(n) = \sum_{j=n}^{j=1} C(n-j+1, n) + \beta n$, where $C(i, n)$ is the cost of summing i intervals each having $n-i+1$ elements
- $C(i, n) = \alpha i(n-i+1)$ (naive)
- $C(i, n) = \alpha((n-i+1) + 2(i-1))$ (clever)

Maximum Sum Subarray by brute force

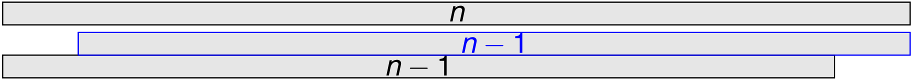
- You are given a one dimensional array that may contain both positive and negative integers
- Find the sum of contiguous subarray of numbers which has the largest sum, e.g.: {-3, -6, **7, -1, -3, 2, 5**, -7}

n

- Time taken is for finding sum of intervals and computing their max
- $T(n) = \sum_{j=n}^{j=1} C(n-j+1, n) + \beta n$, where $C(i, n)$ is the cost of summing i intervals each having $n-i+1$ elements
- $C(i, n) = \alpha i(n-i+1)$ (naive)
- $C(i, n) = \alpha((n-i+1) + 2(i-1))$ (clever)

Maximum Sum Subarray by brute force

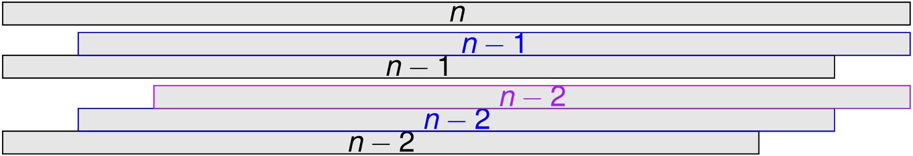
- You are given a one dimensional array that may contain both positive and negative integers
- Find the sum of contiguous subarray of numbers which has the largest sum, e.g.: {-3, -6, **7, -1, -3, 2, 5**, -7}



- Time taken is for finding sum of intervals and computing their max
- $T(n) = \sum_{j=n}^{j=1} C(n-j+1, n) + \beta n$, where $C(i, n)$ is the cost of summing i intervals each having $n-i+1$ elements
- $C(i, n) = \alpha i(n-i+1)$ (naive)
- $C(i, n) = \alpha((n-i+1) + 2(i-1))$ (clever)

Maximum Sum Subarray by brute force

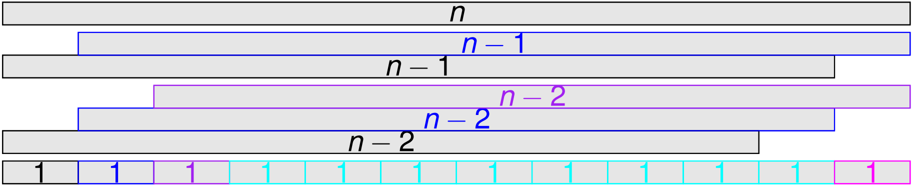
- You are given a one dimensional array that may contain both positive and negative integers
- Find the sum of contiguous subarray of numbers which has the largest sum, e.g.: {-3, -6, **7, -1, -3, 2, 5**, -7}



- Time taken is for finding sum of intervals and computing their max
- $T(n) = \sum_{j=n}^{j=1} C(n-j+1, n) + \beta n$, where $C(i, n)$ is the cost of summing i intervals each having $n-i+1$ elements
- $C(i, n) = \alpha i(n-i+1)$ (naive)
- $C(i, n) = \alpha((n-i+1) + 2(i-1))$ (clever)

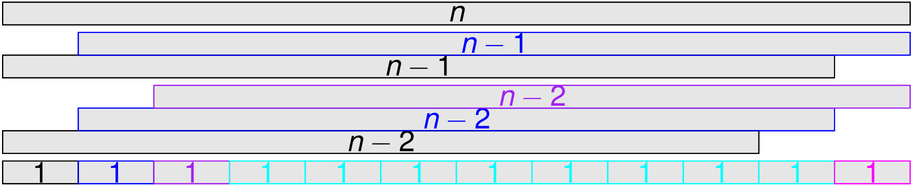
Maximum Sum Subarray by brute force

- You are given a one dimensional array that may contain both positive and negative integers
- Find the sum of contiguous subarray of numbers which has the largest sum, e.g.: {-3, -6, **7, -1, -3, 2, 5**, -7}



Maximum Sum Subarray by brute force

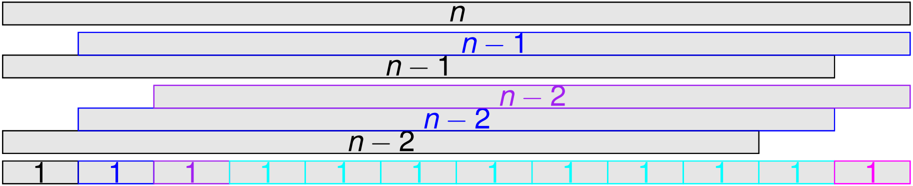
- You are given a one dimensional array that may contain both positive and negative integers
- Find the sum of contiguous subarray of numbers which has the largest sum, e.g.: {-3, -6, **7, -1, -3, 2, 5**, -7}



- Time taken is for finding sum of intervals and computing their max

Maximum Sum Subarray by brute force

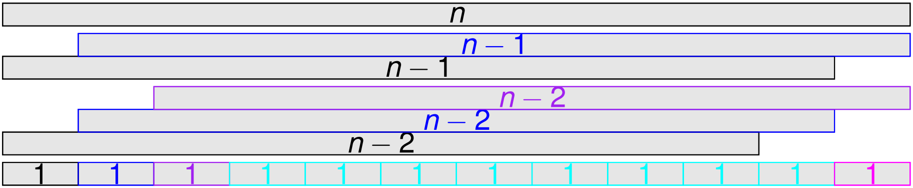
- You are given a one dimensional array that may contain both positive and negative integers
- Find the sum of contiguous subarray of numbers which has the largest sum, e.g.: {-3, -6, **7, -1, -3, 2, 5**, -7}



- Time taken is for finding sum of intervals and computing their max
- $T(n) = \sum_{j=n}^{j=1} C(n-j+1, n) + \beta n$, where $C(i, n)$ is the cost of summing i intervals each having $n-i+1$ elements

Maximum Sum Subarray by brute force

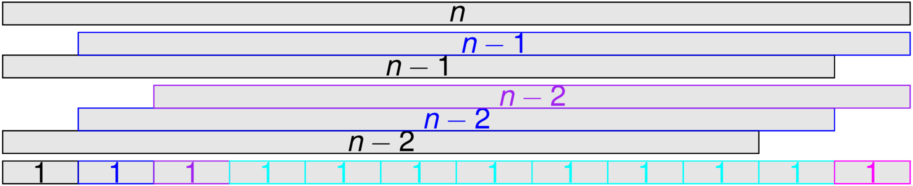
- You are given a one dimensional array that may contain both positive and negative integers
- Find the sum of contiguous subarray of numbers which has the largest sum, e.g.: {-3, -6, **7, -1, -3, 2, 5**, -7}



- Time taken is for finding sum of intervals and computing their max
- $T(n) = \sum_{j=n}^{j=1} C(n-j+1, n) + \beta n$, where $C(i, n)$ is the cost of summing i intervals each having $n-i+1$ elements
- $C(i, n) = \alpha i(n-i+1)$ (naive)

Maximum Sum Subarray by brute force

- You are given a one dimensional array that may contain both positive and negative integers
- Find the sum of contiguous subarray of numbers which has the largest sum, e.g.: {-3, -6, **7, -1, -3, 2, 5**, -7}



- Time taken is for finding sum of intervals and computing their max
- $T(n) = \sum_{j=n}^{j=1} C(n-j+1, n) + \beta n$, where $C(i, n)$ is the cost of summing i intervals each having $n-i+1$ elements
- $C(i, n) = \alpha i(n-i+1)$ (naive)
- $C(i, n) = \alpha((n-i+1) + 2(i-1))$ (clever)

Maximum Sum Subarray by DC

- You are given a one dimensional array that may contain both positive and negative integers
- Find the sum of contiguous subarray of numbers which has the largest sum, e.g.: {-3, -6, **7, -1, -3, 2, 5**, -7}

$$T(n) = \begin{cases} a & n = 1 \\ 2T\left(\frac{n}{2}\right) + bn + c & n > 1 \end{cases}$$

Maximum Sum Subarray by DC

- You are given a one dimensional array that may contain both positive and negative integers
- Find the sum of contiguous subarray of numbers which has the largest sum, e.g.: {-3, -6, **7, -1, -3, 2, 5**, -7}

$$T(n) = \begin{cases} a & n = 1 \\ 2T\left(\frac{n}{2}\right) + bn + c & n > 1 \end{cases}$$

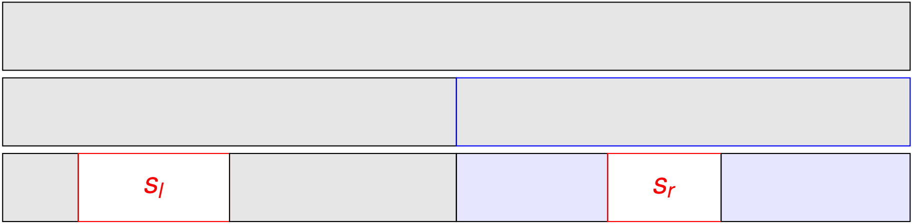
Maximum Sum Subarray by DC

- You are given a one dimensional array that may contain both positive and negative integers
- Find the sum of contiguous subarray of numbers which has the largest sum, e.g.: {-3, -6, **7, -1, -3, 2, 5**, -7}

$$T(n) = \begin{cases} a & n = 1 \\ 2T\left(\frac{n}{2}\right) + bn + c & n > 1 \end{cases}$$

Maximum Sum Subarray by DC

- You are given a one dimensional array that may contain both positive and negative integers
- Find the sum of contiguous subarray of numbers which has the largest sum, e.g.: {-3, -6, **7, -1, -3, 2, 5**, -7}

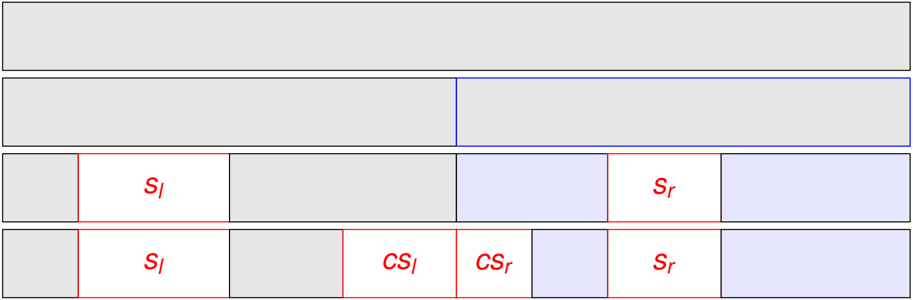


$$T(n) = \begin{cases} a & n = 1 \\ 2T\left(\frac{n}{2}\right) + bn + c & n > 1 \end{cases}$$

$$T(n) \in \Theta(n \lg n)$$

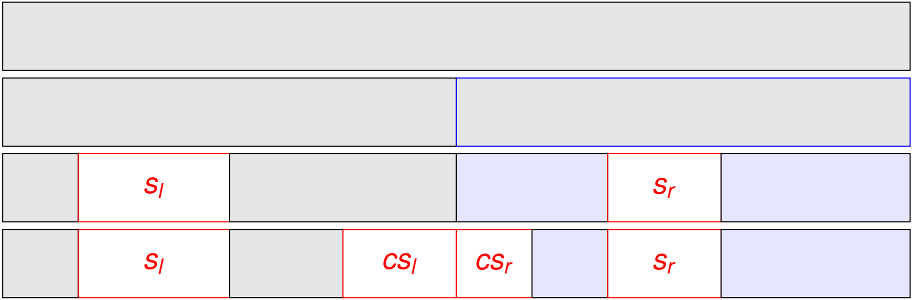
Maximum Sum Subarray by DC

- You are given a one dimensional array that may contain both positive and negative integers
- Find the sum of contiguous subarray of numbers which has the largest sum, e.g.: {-3, -6, **7, -1, -3, 2, 5**, -7}



Maximum Sum Subarray by DC

- You are given a one dimensional array that may contain both positive and negative integers
- Find the sum of contiguous subarray of numbers which has the largest sum, e.g.: {-3, -6, **7, -1, -3, 2, 5**, -7}



$$T(n) = \begin{cases} a & n = 1 \\ 2T\left(\frac{n}{2}\right) + bn + c & n > 1 \end{cases}$$

$$T(n) \in \Theta(n \lg n)$$

Maximum Sum Subarray by DC (contd.)

```
int maxSumSubarray(int A[], int l, int h) {
  // base case when A has only one element
  if (l == h) return A[l];
  // now the recursive steps
  int m = (l + h)/2; // divide A at the middle
  // need to compute max of
  // the maxSumSubarray of each part and
  // the max sum in the interval containing A[m]



}
```

Maximum Sum Subarray by DC (contd.)

```
int maxSumSubarray(int A[], int l, int h) {
  // base case when A has only one element
  if (l == h) return A[l];
  // now the recursive steps
  int m = (l + h)/2; // divide A at the middle
  // need to compute max of
  // the maxSumSubarray of each part and
  // the max sum in the interval containing A[m]
  return max(
    maxSumSubarray(A, l, m),
    maxSumSubarray(A, m+1, h),
    maxCrossSum(A, l, m, h));
}
```

Computing maxCrossSum

```
int maxCrossSum(int A[], int l, int m, int h) {
  // at least A[m] and A[m+1] are present, h>l
  int sum = A[m]; int lSum = A[m];
  for (int i = m-1; i >= l; i--) {
    sum = sum + A[i];
    if (sum > lSum) lSum = sum;
  }
  sum = A[m+1]; int rSum = A[m+1];
  for (int i = m+2; i <= h; i++) {
    sum = sum + A[i];
    if (sum > rSum) rSum = sum;
  }
  return lSum + rSum;
}
```

Matrix multiplication by DC

- Given, $n \times n$ matrices $A = (a_{ij})$, $B = (b_{ij})$, product $C = AB$ is defined as $c_{ij} = \sum_{k=0}^{n} a_{ik} b_{kj}$
- This way, computation of each c_{ij} takes $\Theta(n)$ time
- C is computed in $\Theta(n^3)$ time

Matrix multiplication by DC

- Given, $n \times n$ matrices $A = (a_{ij}), B = (b_{ij})$, product $C = AB$ is defined as $c_{ij} = \sum_{k=0}^{n} a_{ik} b_{kj}$
- This way, computation of each c_{ij} takes $\Theta(n)$ time
- C is computed in $\Theta(n^3)$ time
- Application of DC can be considered
- $A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}$ $B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$
  $C = \begin{pmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{pmatrix}$
- Time complexity: $T(n) =$

Matrix multiplication by DC

- Given, $n \times n$ matrices $A = (a_{ij}), B = (b_{ij})$, product $C = AB$ is defined as $c_{ij} = \sum_{k=0}^{n} a_{ik} b_{kj}$
- This way, computation of each c_{ij} takes $\Theta(n)$ time
- C is computed in $\Theta(n^3)$ time
- Application of DC can be considered
- $A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$
  $C = \begin{pmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{pmatrix}$
- Time complexity: $T(n) = 8T\left(\frac{n}{2}\right) + \Theta(n^2)$
- Solution (by standard methods): $\Theta(n^3)$, so no benefit

Matrix multiplication by DC (contd.)

- Strassen's matrix multiplication
  algorithm works by first defining 7 intermediate matrices as follows:

$$\begin{aligned} D_1 &= (A_{11}+A_{22})(B_{11}+B_{22}) \\ D_2 &= (A_{21}+A_{22})B_{11} \\ D_3 &= A_{11}(B_{12}-B_{22}) \\ D_4 &= A_{22}(B_{21}-B_{11}) \end{aligned}$$

$$\begin{aligned} D_5 &= (A_{11}+A_{12})B_{22} \\ D_6 &= (A_{21}-A_{11})(B_{11}+B_{12}) \\ D_7 &= (A_{12}-A_{22})(B_{21}+B_{22}) \end{aligned}$$

Matrix multiplication by DC (contd.)

- Strassen's matrix multiplication algorithm works by first defining 7 intermediate matrices as follows:

$$\begin{aligned} D_1 &= (A_{11} + A_{22})(B_{11} + B_{22}) \\ D_2 &= (A_{21} + A_{22})B_{11} \\ D_3 &= A_{11}(B_{12} - B_{22}) \\ D_4 &= A_{22}(B_{21} - B_{11}) \end{aligned}$$

$$\begin{aligned} D_5 &= (A_{11} + A_{12})B_{22} \\ D_6 &= (A_{21} - A_{11})(B_{11} + B_{12}) \\ D_7 &= (A_{12} - A_{22})(B_{21} + B_{22}) \end{aligned}$$

- Next, the sub-matrices of C are computed as follows:

$$\begin{aligned} C_{11} &= D_1 + D_4 - D_5 + D_7 & C_{21} &= D_2 + D_4 \\ C_{12} &= D_3 + D_5 & C_{22} &= D_1 - D_2 + D_3 + D_6 \end{aligned}$$

Matrix multiplication by DC (contd.)

- Strassen's matrix multiplication algorithm works by first defining 7 intermediate matrices as follows:

$$\begin{aligned} D_1 &= (A_{11} + A_{22})(B_{11} + B_{22}) \\ D_2 &= (A_{21} + A_{22})B_{11} \\ D_3 &= A_{11}(B_{12} - B_{22}) \\ D_4 &= A_{22}(B_{21} - B_{11}) \end{aligned}$$

$$\begin{aligned} D_5 &= (A_{11} + A_{12})B_{22} \\ D_6 &= (A_{21} - A_{11})(B_{11} + B_{12}) \\ D_7 &= (A_{12} - A_{22})(B_{21} + B_{22}) \end{aligned}$$

- Next, the sub-matrices of C are computed as follows:

$$\begin{aligned} C_{11} &= D_1 + D_4 - D_5 + D_7 & C_{21} &= D_2 + D_4 \\ C_{12} &= D_3 + D_5 & C_{22} &= D_1 - D_2 + D_3 + D_6 \end{aligned}$$

- Time complexity: $T(n) =$

Matrix multiplication by DC (contd.)

- Strassen's matrix multiplication algorithm works by first defining 7 intermediate matrices as follows:

$$\begin{aligned} D_1 &= (A_{11}+A_{22})(B_{11}+B_{22}) \\ D_2 &= (A_{21}+A_{22})B_{11} \\ D_3 &= A_{11}(B_{12}-B_{22}) \\ D_4 &= A_{22}(B_{21}-B_{11}) \end{aligned}$$

$$\begin{aligned} D_5 &= (A_{11}+A_{12})B_{22} \\ D_6 &= (A_{21}-A_{11})(B_{11}+B_{12}) \\ D_7 &= (A_{12}-A_{22})(B_{21}+B_{22}) \end{aligned}$$

- Next, the sub-matrices of C are computed as follows:

$$\begin{aligned} C_{11} &= D_1+D_4-D_5+D_7 & C_{21} &= D_2+D_4 \\ C_{12} &= D_3+D_5 & C_{22} &= D_1-D_2+D_3+D_6 \end{aligned}$$

- Time complexity: $T(n) = 7T\left(\frac{n}{2}\right) + \Theta(n^2)$

Matrix multiplication by DC (contd.)

- Strassen's matrix multiplication
  algorithm works by first defining 7 intermediate matrices as follows:

$$
\begin{aligned}
D_1 &= (A_{11} + A_{22})(B_{11} + B_{22}) \\
D_2 &= (A_{21} + A_{22})B_{11} \\
D_3 &= A_{11}(B_{12} - B_{22}) \\
D_4 &= A_{22}(B_{21} - B_{11})
\end{aligned}
\qquad
\begin{aligned}
D_5 &= (A_{11} + A_{12})B_{22} \\
D_6 &= (A_{21} - A_{11})(B_{11} + B_{12}) \\
D_7 &= (A_{12} - A_{22})(B_{21} + B_{22})
\end{aligned}
$$

- Next, the sub-matrices of C are computed as follows:

$$
\begin{aligned}
C_{11} &= D_1 + D_4 - D_5 + D_7 \\
C_{12} &= D_3 + D_5
\end{aligned}
\qquad
\begin{aligned}
C_{21} &= D_2 + D_4 \\
C_{22} &= D_1 - D_2 + D_3 + D_6
\end{aligned}
$$

- Time complexity: $T(n) = 7T\left(\frac{n}{2}\right) + \Theta(n^2)$
- Solution (by standard methods): $\Theta(n^{2.80735\cdots})$
- Practical utility is limited, but an important result
- Method of Coppersmith and Winograd works in $O(n^{2.376})$
- Trivial lower bound is:

Matrix multiplication by DC (contd.)

- Strassen's matrix multiplication
  algorithm works by first defining 7 intermediate matrices as follows:

$$
\begin{aligned}
D_1 &= (A_{11} + A_{22})(B_{11} + B_{22}) \\
D_2 &= (A_{21} + A_{22})B_{11} \\
D_3 &= A_{11}(B_{12} - B_{22}) \\
D_4 &= A_{22}(B_{21} - B_{11})
\end{aligned}
\qquad
\begin{aligned}
D_5 &= (A_{11} + A_{12})B_{22} \\
D_6 &= (A_{21} - A_{11})(B_{11} + B_{12}) \\
D_7 &= (A_{12} - A_{22})(B_{21} + B_{22})
\end{aligned}
$$

- Next, the sub-matrices of C are computed as follows:

$$
\begin{aligned}
C_{11} &= D_1 + D_4 - D_5 + D_7 \\
C_{12} &= D_3 + D_5
\end{aligned}
\qquad
\begin{aligned}
C_{21} &= D_2 + D_4 \\
C_{22} &= D_1 - D_2 + D_3 + D_6
\end{aligned}
$$

- Time complexity: $T(n) = 7T\left(\frac{n}{2}\right) + \Theta(n^2)$
- Solution (by standard methods): $\Theta(n^{2.80735\cdots})$
- Practical utility is limited, but an important result
- Method of Coppersmith and Winograd works in $O(n^{2.376})$
- Trivial lower bound is: $\Omega(n^2)$

Practice problems

- Given a sorted array in which all elements appear twice (together) and one element appears only once, locate that element
- An array of n points in the plane is given; find out the closest pair of points in the array
- Find the largest rectangular area possible in a given histogram where the largest rectangle can be made of a number of contiguous bars. For simplicity, assume that all bars have same width of 1 unit
- Given a $n \times n$ board where n is a power of 2 with minimum value as 2, with one missing cell (of size 1×1) at a known location, fill the board using L shaped tiles An L shaped tile is a 2×2 square with one cell (of size 1×1) missing
- There are two sorted arrays A and B of size n each, find the median of the array obtained after merging the above 2 arrays
- Check if a given integer p appears more than $\frac{n}{2}$ times in a sorted array of n integers