Introduction to Divide and Conquer Algorithm :

Divide and Conquer нь том асуудлыг жижиг дэд бодлого болгон задалж, дараа нь тэдгээрийн шийдлүүдийг нэгтгэж анхны асуудлыг шийддэг. Үндсэн санаа нь асуудлыг шууд шийдвэрлэхэд хялбар болох хүртэл жижиг дэд бодлогод хуваах явдал юм. Дэд асуудлын шийдлүүдийг олж авсны дараа тэдгээрийг нэгтгэж ерөнхий шийдлийг гаргадаг. Divide and Conquer гурав задалж үзвэл :

1. Divide (Хуваах):
   - Анхны асуудлыг жижиг дэд асуудал болгон хуваа.
   - Дэд асуудал бүр нь ерөнхий асуудлын нэг хэсгийг төлөөлөх ёстой.
   - Зорилго нь асуудлыг цаашид хуваах боломжгүй болтол нь хуваах явдал юм.
2. Conquer(Ялах) : • Жижиг дэд бодлого бүрийг тус тусад нь шийддэг.
   - Хэрэв дэд асуудал хангалттай бага бол үүнийг дахин давталтгүйгээр шууд шийддэг.
   - Зорилго нь эдгээр дэд асуудлын шийдлийг бие даан олох явдал юм.
3. Merge(Нэгтгэх) :
   - Бүх асуудлын эцсийн шийдлийг гаргахын тулд дэд асуудлуудыг нэгтгэнэ.
   - Жижиг дэд асуудлууд шийдэгдсэний дараа бид илүү том асуудлын шийдлийг олж авахын тулд тэдгээрийн шийдлүүдийг рекурсив байдлаар нэгтгэдэг.
   - Зорилго нь дэд асуудлын үр дүнг нэгтгэх замаар анхны асуудлын шийдлийг олох явдал юм.

   Жишээ нь : Хэрэв бид [38, 27, 43, 3, 9, 82, 10] гэх мэт эрэмбэлэгдээгүй массивтай бол :

   1. Массивыг [38, 27, 43, 3] болон [9, 82, 10] гэх мэтчилэн нэг элемент бүхий жижиг хэсгүүд рүү хуваана.
   2. Дараа нь эдгээрийг эрэмбэлж, нэгтгэн бүрэн эрэмбэлэгдсэн массив болох [3, 9, 10, 27, 38, 43, 82] гаргаж авна.

Pseudo code :

```
def merge_sort(arr):
    if len(arr) <= 1:
        return arr

    mid = len(arr) // 2
    left_half = merge_sort(arr[:mid])
    right_half = merge_sort(arr[mid:])

    return merge(left_half, right_half)

def merge(left, right):
    sorted_array = []
    i = j = 0

    while i < len(left) and j < len(right):
        if left[i] < right[j]:
            sorted_array.append(left[i])
            i += 1
        else:
            sorted_array.append(right[j])
            j += 1

    sorted_array.extend(left[i:])
    sorted_array.extend(right[j:])

    return sorted_array
```

- **Dynamic programming(DP)** :

  - DP нь тооцооллын шинжлэх ухаан болон математикийг ашиглан нарийн төвөгтэй асуудлыг бага давхацсан дэд асуудлуудад хуваах замаар шийдвэрлэх алгоритмын арга юм.

  - Dynamic programming-ийн үндсэн санаа нь дэд асуудлуудын шийдлийг хадгалах бөгөөд ингэснээр тус бүрийг зөвхөн нэг удаа л шийддэг.

  - DP асуудлыг шийдэхийн тулд бид эхлээд дахин давтагдсан дэд асуудлууд бүхий нөхцөлд рекурсив шийдлийг бичдэг (рекурсив функц нь нэг параметрээр олон удаа дуудаг).

- Үр дүнг хадгалах хоёр арга бий, эхнийх нь дээрээс доош (memorization) ба нөгөө нь доороос дээш (tabulation) юм.

  Жишээ нь: Fibonacci.
  Фибоначчийн тоо нь өмнөх хоёр тооны нийлбэрээр тодорхойлогддог дараалал юм.

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

```python
def fibonacci(n):
    if n <= 1:
        return n
    return fibonacci(n-1) + fibonacci(n-2)
```

- **Хомхойлох Алгоритм (Greedy Algorithm) :**

  Хомхойлох алгоритм нь асуудлыг шийдвэрлэхдээ тухайн үеийн хамгийн сайн, хамгийн их ашиг тустай сонголтыг сонгож явдаг алгоритмын нэг арга юм. Хомхойлох алгоритмын хувьд тухайн мөчийн хамгийн оновчтой сонголт ирээдүйн хамгийн оновчтой шийдэлд хүргэнэ гэж үздэг билээ.

  Жишээ Бодлого: Зоосны асуудал (Coin Change Problem)

  63₮-ийн дүнг дараах дэвсгэртүүдийн тусламжтайгаар (1₮, 5₮, 10₮, 20₮, 50₮) төлөхөөр ажиллая.

```python
def min_coins(amount, coins):
    coins.sort(reverse=True)
    count = 0
    result = []

    for coin in coins:
        while amount >= coin:
            amount -= coin
            count += 1
            result.append(coin)

    return count, result
```

Бодох нь:

63₮-ийн дүнтэй үед хамгийн том зоос болох 50₮-ийг эхэлж авна:

1. 63₮ - 50₮ = 13₮.
2. Дараагийн том зоос болох 10₮-г ашиглана: 13₮ - 10₮ = 3₮.

3. Үлдсэн 3₮-д 1₮ зоосыг гурван удаа ашиглана.

2-р хэсэг.

- **Recursion vs Divide-and-Conquer .**

  Recursion нь дангаараа асуудлыг жижгэрүүлэн, үндсэн тохиолдол руу хүрэх явцад ашиглагддаг бөгөөд дахин дуудлага ашиглан шийддэг.

  Жишээ нь : factorial and fibonacci

  Divide-and-Conquer нь Recursion дээр суурилсан хэдий ч асуудлыг дахин хувааж, жижиг дэд асуудлуудыг тус тусад нь бие даан шийдэж, эцэст нь нэгтгэж шийддэг.

  Жишээ нь : Merge Sort, Quick Sort, Binary Search

Recursion бол өөрөө өөрийгөө дуудаж асуудлыг энгийн болгон хялбаршуулдаг *арга* бол, Divide-and-Conquer нь Recursion дээр суурилан, дэд асуудлуудыг нэгтгэхэд анхаарч, илүү үр ашигтай шийдэл гаргах *арга* юм.

- **Divide-and-Conquer vs Dynamic Programming .**

Divide-and-Conquer:

Асуудлыг жижиг хэсгүүдэд хуваана. Хуваасан асуудлуудыг тус тусад нь шийднэ. Түүний дараа шийдлийг нэгтгэж анхны асуудлын хариуг олно.

Жишээ нь : Merge Sort алгоритм нь массивыг хоёр хэсэгт хувааж, тус бүрийг эрэмбэлээд эцэст нь нэгтгэж, эрэмбэлсэн массивыг гаргадаг.

Dynamic Programming:

Давхцсан дэд асуудлуудыг олж, тооцоолол үүсгэдэг ба үүний дараа шийдлийг санах ойд хадгалдаг. Мөн хадгалсан утгуудыг дахин ашиглах замаар ажилладаг.

Жишээ нь : fibonacci дарааллын тоонуудыг олоход давхацсан дэд тооцооллууд үүсдэг.

Divide-and-Conquer нь асуудлыг томоос жижгэрүүлэн хуваах замаар шийддэг бол, Dynamic Programming нь дахин ашиглах боломжтой давхацсан дэд асуудлуудыг ашиглан тооцооллын үр ашгийг нэмэгдүүлдэг.

- **Dynamic Programming vs Greedy .**

DP нь бүх боломжийг тооцон хамгийн оновчтой шийдлийг олдог тул, ихэвчлэн илүү найдвартай, нарийвчлалтай хариу гаргадаг.

Greedy алгоритм нь хурдтай бөгөөд хялбар боловч, заримдаа хамгийн оновчтой шийдэлд хүрч чадахгүй байж болдог.

Ашигласан материал :

https://www.geeksforgeeks.org/introduction-to-divide-and-conquer-algorithm/

https://www.geeksforgeeks.org/dynamic-programming/ https://www.geeksforgeeks.org/introduction-togreedy-algorithm-data-structures-and-algorithm-tutorials/