ШИНЖЛЭХ УХААН ТЕХНОЛОГИЙН ИХ СУРГУУЛЬ

Мэдээлэл холбооны технологийн сургууль

Бие даалт 2

Алгоритмын шинжилгээ ба зохиомж (F.CS301)

2024-2025 оны хичээлийн жилийн намар

1. Divide-and-Conquer
   Divide-and-Conquer нь асуудлыг жижиг хэсгүүдэд хувааж, тус бүрийг нь шийдээд, үр дүнг нь нэгтгэн төгс шийдлийг олох арга.
   Жишээ: Merge Sort

```python
def merge_sort(arr):
    if len(arr) <= 1:
        return arr
    mid = len(arr) // 2
    left = merge_sort(arr[:mid])
    right = merge_sort(arr[mid:])

    return merge(left, right)

def merge(left, right):
    result = []
    i = j = 0
    while i < len(left) and j < len(right):
        if left[i] < right[j]:
            result.append(left[i])
            i += 1
        else:
            result.append(right[j])
            j += 1

    result.extend(left[i:])
    result.extend(right[j:])
    return result

# Жишээ ашиглах
arr = [38, 27, 43, 3, 9, 82, 10]
print(merge_sort(arr))  # [3, 9, 10, 27, 38, 43, 82]
```

2. Dynamic Programming
   Dynamic Programming нь нэгэн ижил төстэй дэд асуудлуудыг дахин дахин шийдэхээс сэргийлэхэд ашигладаг арга юм. Өөрөөр хэлбэл, энэ нь дахин тооцохгүйгээр нэг удаа шийдсэн дэд асуудлын үр дүнг хадгалж ашиглана.
   Жишээ: Fibonacci тоонууд

```python
def fibonacci(n, memo={}):
    if n <= 1:
        return n
    if n not in memo:
        memo[n] = fibonacci(n-1, memo) + fibonacci(n-2, memo)
    return memo[n]

# Жишээ ашиглах
print(fibonacci(10))  # 55
```

3. Greedy Algorithms
   Greedy алгоритм нь тухайн үед хамгийн сайн шийдлийг сонгох бөгөөд энэ нь урьдчилан харагдах хамгийн сайн сонголт гэж үздэг. Гэвч энэ нь бүх үед хамгийн оновчтой шийдлийг өгч чадахгүй.
   Жишээ: Coin Change

```python
def coin_change(coins, amount):
    coins.sort(reverse=True)
    count = 0
    for coin in coins:
        count += amount // coin
        amount = amount % coin
    return count

# Жишээ ашиглах
coins = [25, 10, 5, 1]
amount = 63
print(coin_change(coins, amount))  # 6
```

Харьцуулалт

1. Recursion vs Divide-and-Conquer
   Recursion функц нь өөрийгөө дууддаг арга бөгөөд, өөрийнхөө дэд асуудлыг шийдэж, шийдвэрийн үр дүнг буцаадаг арга бол
   Divide-and-Conquer нь тухайн асуудлыг жижиг хэсгүүдэд хувааж, тус бүрийг нь шийдээд, үр дүнг нь нэгтгэн шийдлийг гаргадаг.

```python
# Рекурсийн аргаар Factorial тооцох
def factorial_recursive(n):
    if n == 0 or n == 1:
        return 1
    else:
        return n * factorial_recursive(n - 1)

# Жишээ ашиглах
n = 5
print(f"Factorial of {n} using recursion: {factorial_recursive(n)}")  # 120


# Divide-and-Conquer аргаар Factorial тооцох
def factorial_divide_conquer(n):
    if n == 0 or n == 1:
        return 1
    mid = n // 2
    left_result = factorial_divide_conquer(mid)  # Хоёрдугаар хэсэг
    right_result = factorial_divide_conquer(n - mid)  # Нөгөө хэсэг
    return left_result * right_result * n  # Хэсгүүдийг нэгтгэж үр дүн гаргана

# Жишээ ашиглах
n = 5
print(f"Factorial of {n} using divide-and-conquer: {factorial_divide_conquer(n)}")
```

Гол ялгаа:
Recursion нь зөвхөн нэг дэд асуудлыг шийдэж дуусахаас өмнө буцаж ирдэг.
Divide-and-Conquer нь олон дэд асуудлыг хувааж шийдэж, эдгээрийг нэгтгэн эцсийн шийдлийг гаргадаг.

2. Divide-and-Conquer vs Dynamic Programming
Divide-and-Conquer нь хувааж, тус бүрийг шийдээд, нэгтгэдэг, гэхдээ энэ нь дэд асуудлыг дахин дахин шийдэж болно.
Dynamic Programming нь дахин давтагдсан дэд асуудлуудыг хадгалаад, дахин тооцоолохоос сэргийлж, нэг удаа шийдсэн дэд асуудлуудын үр дүнг хадгалан ашигладаг.

```python
# Divide-and-Conquer аргаар Fibonacci тооцох
def fibonacci_divide_conquer(n):
    if n <= 1:
        return n
    else:
        return fibonacci_divide_conquer(n-1) + fibonacci_divide_conquer(n-2)

# Жишээ ашиглах
n = 10
print(f"Fibonacci of {n} using divide-and-conquer: {fibonacci_divide_conquer(n)}")

# Dynamic Programming аргаар Fibonacci тооцох
def fibonacci_dp(n, memo={}):
    if n <= 1:
        return n
    if n not in memo:
        memo[n] = fibonacci_dp(n-1, memo) + fibonacci_dp(n-2, memo)
    return memo[n]

# Жишээ ашиглах
n = 10
print(f"Fibonacci of {n} using dynamic programming: {fibonacci_dp(n)}")  # 55
```

Гол ялгаа:
Divide-and-Conquer нь бие даан шийдэгдсэн дэд асуудлуудын үр дүнг давтан тооцоолдог, харин Dynamic Programming нь эдгээр үр дүнг хадгалж, дахин тооцоолохоос зайлсхийдэг.

3. Dynamic Programming vs Greedy
Dynamic Programming нь бүх боломжит шийдлийг хадгалж, хамгийн оновчтой шийдлийг олохыг зорьдог.
Greedy нь хамгийн сайн шийдлийг сонгон авч, бүх боломжит шийдлүүдийг бодолцохгүйгээр шууд шийдлийг гаргадаг.

```python
# Dynamic Programming аргаар Coin Change
def coin_change_dp(coins, amount):
    dp = [float('inf')] * (amount + 1)
    dp[0] = 0

    for i in range(1, amount + 1):
        for coin in coins:
            if coin <= i:
                dp[i] = min(dp[i], dp[i - coin] + 1)

    return dp[amount] if dp[amount] != float('inf') else -1

# Жишээ ашиглах
coins = [1, 5, 10, 25]
amount = 63
print(f"Minimum coins needed (DP): {coin_change_dp(coins, amount)}")  # 6

# Greedy аргаар Coin Change
def coin_change_greedy(coins, amount):
    coins.sort(reverse=True)
    count = 0
    for coin in coins:
        if amount == 0:
            break
        count += amount // coin  # Хамгийн том coin сонгож, дүнг хасна
        amount = amount % coin   # Үлдсэн мөнгөний дүнг тооцно

    return count if amount == 0 else -1

# Жишээ ашиглах
coins = [1, 5, 10, 25]
amount = 63
print(f"Minimum coins needed (Greedy): {coin_change_greedy(coins, amount)}")
```

Гол ялгаа:
DP нь хамгийн сайн шийдлийг олдог, гэхдээ илүү их цаг санах ой шаарддаг.
Greedy нь хурдан бөгөөд хялбар, гэхдээ бүх тохиолдолд зөв шийдэл гарахгүй.