

목차

1. Implement Bin Packing Scheduling Algorithms for the Next Fit, First Fit, Best Fit and Worst Fit.	2
Common main function (main function baseline code).....	2
Next Fit	2
First Fit.....	2
Best Fit	3
Worst Fit.....	4
2. Analyze the time and space complexity on each algorithm.	5
Next Fit	5
First Fit.....	5
Best Fit	6
Worst Fit.....	6
3. Impressions.....	7

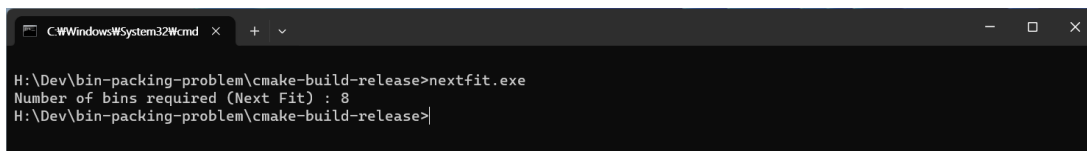
1. Implement Bin Packing Scheduling Algorithms for the Next Fit, First Fit, Best Fit and Worst Fit.

Common main function (main function baseline code)

```
int main() {
    int weight[] = {2, 5, 4, 7, 1, 3, 8, 2, 5, 4, 7, 1, 3, 8};
    int c = 10;
    int n = sizeof(weight) / sizeof(weight[0]);
    cout << "Number of bins required (Target Fit) : "
          << target_fit_function(weight, n, c);
    return 0;
}
```

Next Fit

```
// Next Fit 알고리즘을 이용하여 필요한 bin 의 개수를 반환
int next_fit(int weight[], int n, int c) {
    // 결과 (bin 수)와 현재 bin 의 남은 용량을 초기화
    int result = 0;
    int bin_remain = c;
    // 항목을 하나씩 배치
    for (int i = 0; i < n; i++) {
        // 이 항목이 현재 bin 에 들어갈 수 없다면
        if (weight[i] > bin_remain) {
            result++;
            // 새로운 bin 을 사용
            bin_remain = c - weight[i];
        } else
            bin_remain -= weight[i];
    }
    return result;
}
```



```
C:\Windows\System32\cmd - + v
H:\Dev\bin-packing-problem\cmake-build-release>nextfit.exe
Number of bins required (Next Fit) : 8
H:\Dev\bin-packing-problem\cmake-build-release>
```

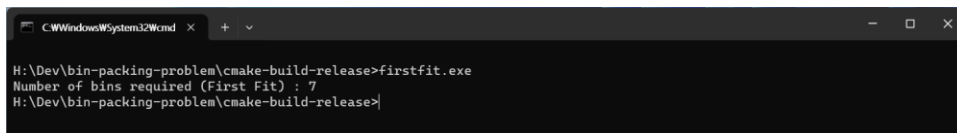
First Fit

```
// First Fit 알고리즘을 이용하여 필요한 bin 의 개수를 반환
int first_fit(const int weight[], int n, int c) {
    // bin 개수를 초기화
    int result = 0;
    // bin 에서 남은 공간을 저장하기 위한 배열을 생성
    int *bin_remain = new int[n];
```

```

// 항목을 하나씩 배치
for (int i = 0; i < n; i++) {
    // weight[i]를 넣을 수 있는 첫 번째 bin을 찾기
    int j;
    for (j = 0; j < result; j++) {
        if (bin_remain[j] >= weight[i]) {
            bin_remain[j] = bin_remain[j] -
weight[i];
            break;
        }
    }
    // weight[i]를 수용할 수 있는 bin이 없다면
    if (j == result) {
        bin_remain[result] = c - weight[i];
        result++;
    }
}
delete[] bin_remain;
return result;
}

```



```

C:\Windows\System32\cmd
H:\Dev\bin-packing-problem\cmake-build-release>firstfit.exe
Number of bins required (First Fit) : 7
H:\Dev\bin-packing-problem\cmake-build-release>

```

Best Fit

```

// Best Fit 알고리즘을 이용하여 필요한 bin의 개수를 반환
int best_fit(int weight[], int n, int c) {
    // bin 개수를 초기화
    int result = 0;
    // bin에서 남은 공간을 저장하기 위한 배열을 생성
    int *bin_remain = new int[n];
    // 항목을 하나씩 배치
    for (int i = 0; i < n; i++) {
        // weight[i]를 넣을 수 있는 최적 bin을 찾기
        int j;
        // 남은 최소 공간과 최적 bin의 색인을 초기화
        int min = c + 1, bin_index = 0;
        for (j = 0; j < result; j++) {
            if (bin_remain[j] >= weight[i] &&
bin_remain[j] - weight[i] < min) {
                bin_index = j;
                min = bin_remain[j] - weight[i];
            }
        }
        // weight[i]를 넣을 수 있는 bin이 없다면 새로운 bin을
        생성
        if (min == c + 1) {

```

```

        bin_remain[result] = c - weight[i];
        result++;
    } else {
        // 항목을 최적의 bin 에 배정
        bin_remain[bin_index] -= weight[i];
    }
}
delete[] bin_remain;
return result;
}

```

```

C:\Windows\System32\cmd
H:\Dev\bin-packing-problem\cmake-build-release>bestfit.exe
Number of bins required (Best Fit) : 7
H:\Dev\bin-packing-problem\cmake-build-release>

```

Worst Fit

```

// Worst Fit 알고리즘을 이용하여 필요한 bin 의 개수를 반환
int worst_fit(int weight[], int n, int c) {
    // bin 개수를 초기화
    int result = 0;
    // bin 에서 남은 공간을 저장하기 위한 배열을 생성
    int *bin_remain = new int[n];
    // 항목들을 하나씩 배치
    for (int i = 0; i < n; i++) {
        // weight[i]를 넣을 수 있는 최적의 bin 를 찾기
        int j;
        // 남은 최대 공간과 가장 적합하지 않은 bin 의 인덱스를
        초기화
        int max = -1, worst_index = 0;
        for (j = 0; j < result; j++) {
            if (bin_remain[j] >= weight[i] &&
                bin_remain[j] - weight[i] > max) {
                worst_index = j;
                max = bin_remain[j] - weight[i];
            }
        }
        // weight[i]를 넣을 수 있는 bin 이 없다면 새로운 bin 을
        생성
        if (max == -1) {
            bin_remain[result] = c - weight[i];
            result++;
        } else {
            // 항목을 가장 적합한 bin 에 배정
            bin_remain[worst_index] -= weight[i];
        }
    }
    delete[] bin_remain;
    return result;
}

```

```
C:\Windows\System32\cmd x + v
H:\Dev\bin-packing-problem\cmake-build-release>worstfit.exe
Number of bins required (Worst Fit) : 7
H:\Dev\bin-packing-problem\cmake-build-release>
```

2. Analyze the time and space complexity on each algorithm.

Next Fit

```
for (int i = 0; i < n; i++) {
    // 이 항목이 현재 bin에 들어갈 수 없다면
    if (weight[i] > bin_remain) {
        result++;
        // 새로운 bin을 사용
        bin_remain = c - weight[i];
    } else
        bin_remain -= weight[i];
}
```

[시간 복잡도] next_fit 함수 내부의 주요 연산은 weight들을 하나씩 반복하면서 현재의 bin에 넣는 것이다. (넣지 못하는 경우 새로운 bin을 사용) 이 연산은 각 weight에 대해서 수행되므로 시간 복잡도는 $O(n)$ 이다.

[공간 복잡도] 주어진 입력에 기반한 weight 배열이 $O(n)$ 의 공간을, bin_remain 및 result는 상수 공간을 차지하므로 $O(1)$ 의 공간을 차지한다. 따라서 전체 공간 복잡도는 $O(n) + O(1) = O(n)$ 이다.

First Fit

```
for (int i = 0; i < n; i++) {
    // weight[i]를 넣을 수 있는 첫 번째 bin를 찾기
    int j;
    for (j = 0; j < result; j++) {
        if (bin_remain[j] >= weight[i]) {
            bin_remain[j] = bin_remain[j] - weight[i];
            break;
        }
    }
    // weight[i]를 수용할 수 있는 bin이 없다면
    if (j == result) {
        bin_remain[result] = c - weight[i];
        result++;
    }
}
```

[시간 복잡도] first_fit 함수 내부의 주요 연산은 weight들을 하나씩 반복하면서 weight를 넣을 수 있는 bin을 전체 bin에서 찾는 것이다. 이 연산은 각 weight와 result ($result \leq n$)에 대해서 중첩 루프의 형태로 실행되므로 전체 시간 복잡도는 $O(n^2)$ 이다.

[공간 복잡도] 주어진 입력에 기반한 weight 배열과 bin_remain 배열이 $O(n)$ 의 공간을 차지하고, result, c, j 등의 변수는 상수 공간을 차지하므로 $O(1)$ 의 공간을 차지한다. 따라서

전체 공간 복잡도는 $O(n) + O(n) + O(1) = O(n)$ 이다.

Best Fit

```
for (int i = 0; i < n; i++) {
    // weight[i]를 넣을 수 있는 최적 bin을 찾기
    int j;
    // 남은 최소 공간과 최적 bin의 색인을 초기화
    int min = c + 1, bin_index = 0;
    for (j = 0; j < result; j++) {
        if (bin_remain[j] >= weight[i] && bin_remain[j]
- weight[i] < min) {
            bin_index = j;
            min = bin_remain[j] - weight[i];
        }
    }
    // weight[i]를 넣을 수 있는 bin이 없다면 새로운 bin을
    생성
    if (min == c + 1) {
        bin_remain[result] = c - weight[i];
        result++;
    } else {
        // 항목을 최적의 bin에 배정
        bin_remain[bin_index] -= weight[i];
    }
}
```

[시간 복잡도] best_fit 함수 내부의 주요 연산은 weight들을 하나씩 반복하면서 전체 bin 중에 가장 남은 공간이 적으면서 weight보다는 큰 bin을 찾아 weight를 넣는 것이다. 이 연산은 각 weight와 result ($result \leq n$)에 대해서 중첩 루프의 형태로 실행되므로 전체 시간 복잡도는 $O(n^2)$ 이다.

[공간 복잡도] 주어진 입력에 기반한 weight 배열과 bin_remain 배열이 $O(n)$ 의 공간을 차지하고 result, c, j, min, bin_index 등의 변수는 상수 공간을 차지하므로 $O(1)$ 의 공간을 차지한다. 따라서 전체 공간 복잡도는 $O(n) + O(n) + O(1) = O(n)$ 이다.

Worst Fit

```
for (int i = 0; i < n; i++) {
    // weight[i]를 넣을 수 있는 최적의 bin를 찾기
    int j;
    // 남은 최대 공간과 가장 적합하지 않은 bin의 인덱스를 초기화
    int max = -1, worst_index = 0;
    for (j = 0; j < result; j++) {
        if (bin_remain[j] >= weight[i] && bin_remain[j]
- weight[i] > max) {
            worst_index = j;
        }
    }
}
```

```

        max = bin_remain[j] - weight[i];
    }
}
// weight[i]를 넣을 수 있는 bin 이 없다면 새로운 bin 을
생성
if (max == -1) {
    bin_remain[result] = c - weight[i];
    result++;
} else {
    // 항목을 가장 적합한 bin 에 배정
    bin_remain[worst_index] -= weight[i];
}
}

```

[시간 복잡도] worst_fit 함수 내부의 주요 연산은 weight들을 하나씩 반복하면서 전체 bin 중에 가장 남은 공간이 큰 bin을 찾아 weight를 넣는 것이다. 이 연산은 각 weight와 result ($result \leq n$)에 대해서 중첩 루프의 형태로 실행되므로 전체 시간 복잡도는 $O(n^2)$ 이다.

[공간 복잡도] 주어진 입력에 기반한 weight 배열과 bin_remain 배열이 $O(n)$ 의 공간을 차지하고 result, c, j, max, worst_index 등의 변수는 상수 공간을 차지하므로 $O(1)$ 의 공간을 차지한다. 따라서 전체 공간 복잡도는 $O(n) + O(n) + O(1) = O(n)$ 이다.

3. Impressions

상자 채우기 문제(Bin Packing Problem)을 해결할 수 있는 여러가지 각 방법을 직접 구현해보고, 각 상황에서의 시간 복잡도와 공간 복잡도를 구해보면서, 예전에 배웠던 개념들을 다시 복습해볼 수 있는 시간을 가질 수 있어서 좋았다. 특히 본 문제 상황 같은 경우 상자를 곧 프로세서 하나로 생각하면, 멀티 프로세싱 환경에서의 스케줄링에 사용될 수 있을 것 같다는 생각이 들었다.