```
/Users/it/Desktop/os/labs/untitled3/cmake-build-debug/untitled3
Enter the number of jobs: 5
Enter size for process 1: 200
Enter size for process 2: 100
Enter size for process 3: 300
Enter size for process 4: 300
Enter size for process 5: 200

Enter the number of Partitions: 4
Enter size for partition 1: 100
Enter size for partition 2: 300
Enter size for partition 3: 200
Enter size for partition 4: 450
-----------------------------------------------------
```

```
-----------------------------------------------------------

        First Fit Algorithm
JOB ID      Partition ID    Waste       Status
1       2       100     running
2       1       0       running
3       4       150     running
4       -       -       waiting
5       3       0       running
Total Waste: 250

-----------------------------------------------------------
```

```
------------------------------------------------------------

        Next Fit Algorithm
JOB ID      Partition ID    Waste       Status
1       2       100     running
2       3       100     running
3       4       150     running
4       -       -       waiting
5       -       -       waiting
Total Waste: 350

------------------------------------------------------------
```

```
------------------------------------------------------------

        Best Fit Algorithm
JOB ID      Partition ID    Waste       Status
1       3       0       running
2       1       0       running
3       2       0       running
4       4       150     running
5       -       -       waiting
Total Waste: 150

------------------------------------------------------------
```

```
------------------------------------------------------------

        Worst Fit (Fixed) Algorithm
JOB ID      Partition ID    Waste       Status
1       4       250     running
2       2       200     running
3       -       -       waiting
4       -       -       waiting
5       3       0       running
Total Waste: 450

------------------------------------------------------------
```

```
------------------------------------------------------------

        Worst Fit (DYNAMIC) Algorithm
JOB ID      Partition ID    Waste       Status
1       4       250     running
2       2       200     running
3       -       -       waiting
4       -       -       waiting
5       4.1     50      running
Total Waste: 0
```

```
      ------- Main Memory Configuration -------
Partitions      Partition Size      Job Assigned
1               100                 -
2               100                 2
2.1             200                 -
3               200                 -
4               200                 1
4.1             200                 5
4.2             50                  -

Process finished with exit code 0
```

```cpp
#include <iostream>
#include <string>
#include <vector>
#include <iomanip>

using namespace std;
```

```
vector<class Job> setJob(); //
declaring a vector of Job objects
vector<class Partition>
setPartition(); //declaring a vector
of Partition objects
void print(vector<class Job> jobList,
vector<class Partition> partitionList,
int totalWaste); //function to print
out information
void FirstFit(vector<class Job>,
vector<class Partition>);  //first fit
function
void NextFit(vector<class Job>,
vector<class Partition>);  //NextFit
function
void BestFit(vector<class Job>,
vector<class Partition>); //bestFit
function
```

```
void WorstFitFixed(vector<class Job>, vector<class Partition>);  //WorstFit (fixed) function
void WorstFitDynamic(vector<class Job>, vector<class Partition>); //WorstFit (dynamic) function

class Job //Job class
{
public:
    Job() { process_id = -1, process_Size = 0, status = false, partitionNum = -1; } //default constructor for Job object
    Job(int n, int size) //constructor for Job object
    {
        process_id = n;
        process_Size = size;
```

```
    }
    void setProcess_id(int n) //
function for setting process id
    {
        process_id = n;
    }
    void setSize(int size) //function
for setting size
    {
        process_Size = size;
    }
    void setStatus(bool s) //function
for setting status
    {
        status = s;
    }
    void setPartitionNum(double p) //
function for setting partition number
```

```
    {
        partitionNum = p;
    }
    void setWaste(int w) //function
for setting waste
    {
        waste = w;
    }
    int getProcess_id() //function for
getting process id
    {
        return process_id;
    }
    int getSize() //function for
getting size
    {
        return process_Size;
    }
```

```
    bool getStatus() //function for
getting status
    {
        return status;
    }

    double getPartNum() //function for
getting partition number
    {
        return partitionNum;
    }

    int getWaste() //function for
getting waste
    {
        return waste;
    }
```

```
private: //private variables for the
Job class
    int process_id;
    int process_Size;
    bool status = false;
    double partitionNum = -1;
    int waste = -1;
};

class Partition //Partition class
{
public:
    Partition() { partition_id = -1,
size = 0, assigned = false, process_ID
= -1, waste = -1; } //default
constructor for Partition object
    Partition(double n, int s) //
constructor for Partition object
    {
```

```
        partition_id = n;
        size = s;
        assigned = false;
        waste = -1;
        process_ID = -1;
    }
    void setPartition_id(double n) //
function for setting partition id
    {
        partition_id = n;
    }
    void setSize(int s) // function
for setting size
    {
        size = s;
    }
```

```
    void setAssign(bool a) // function
for setting whether or not the
partition was assigned or not (status)
    {
        assigned = a;
    }
    void setprocess_ID(int j) //
function for setting process id
    {
        process_ID = j;
    }
    void setWaste(int w) // function
for setting waste
    {
        waste = w;
    }
    double getPartition_id() //
function for getting partition id
    {
```

```
        return partition_id;
    }
    int getSize() // function for
getting size
    {
        return size;
    }
    int getAssigned() // function for
getting the assigned bool
    {
        return assigned;
    }
    int getprocess_ID() // function
for getting process id
    {
        return process_ID;
    }
```

```
    int getWaste() // function for
getting waste
    {
        return waste;
    }

private: //private variables for
Partition class
    double partition_id = -1;
    int size = -1;
    bool assigned = false;
    int process_ID = -1;
    int waste = -1;
};

int main()
{
```

```
    vector<Job> jobList; //creating a
vector of Job objects called jobList
    vector<Partition>
partitionList; //creating a vector of
Partition objects called partitionList
    jobList = setJob(); //setting the
vector to a setJob function that
inputs values
    partitionList = setPartition(); //
setting the vector to a setPartition
function that inputs values
    FirstFit(jobList,
partitionList); //calling the
algorithm function that takes in both
vectors as parameters
    NextFit(jobList,
partitionList); //calling the
algorithm function that takes in both
vectors as parameters
```

```
    BestFit(jobList,
partitionList); //calling the
algorithm function that takes in both
vectors as parameters
    WorstFitFixed(jobList,
partitionList); //calling the
algorithm function that takes in both
vectors as parameters
    WorstFitDynamic(jobList,
partitionList); //calling the
algorithm function that takes in both
vectors as parameters

    return 0;
}

void print(vector<Job> jobList,
vector<Partition> partitionList, int
totalWaste) //print function that
```

```
outputs the information related to a
algorithm
{
    cout << "JOB ID"
         << "\t\tPartition ID"
         << "\tWaste"
         << "\t\tStatus" << endl;
    string yesNo;
    for (int i = 0; i <
jobList.size(); i++)
    {
        if (jobList[i].getStatus()) //
printing out the status of a process
        {
            yesNo = "running";
        }
        else
            yesNo = "waiting";
```

```
        int x = jobList[i].getWaste();
//variable to represent waste
        if (x == -1)
        {
            //printing out a dash if
the value is negative
            cout <<
jobList[i].getProcess_id() << "\t\t"
                 << "-"
                 << "\t\t"
                 << "-"
                 << "\t\t" << yesNo <<
endl;
        }
        else
        {
            cout <<
jobList[i].getProcess_id() << "\t\t"
```

```
<< jobList[i].getPartNum() << "\t\t"
<< jobList[i].getWaste() << "\t\t" <<
yesNo << endl; //printing out
information
        }
    }
    cout << "Total Waste: " <<
totalWaste << endl
         << endl;
}
vector<Job> setJob() //setJob function
that gets information relating to the
Job
{
    vector<Job> jobList;
    int numofJobs, sizeOfJob;
    cout << "Enter the number of jobs:
"; //asking for unmber of jobs
    cin >> numofJobs;
```

```
    for (int i = 1; i <= numofJobs; i++)
    {
        cout << "Enter size for process " << i << ": "; //entering the size of each process
        cin >> sizeOfJob;
        jobList.push_back(Job(i, sizeOfJob)); // adding the information to the vector
    }
    cout << endl;
    return jobList;
}
vector<Partition> setPartition() // function for entering information about the partition
{
    vector<Partition> partitionList;
```

```
    int numOfPart, sizeOfPart;
    // Setting Up Partitions
    cout << "Enter the number of
Partitions: "; //entering the number
of partitions
    cin >> numOfPart;
    for (int i = 1; i <= numOfPart; i+
+)
    {
        cout << "Enter size for
partition " << i << ": ";

        cin >> sizeOfPart;

partitionList.push_back(Partition(i,
sizeOfPart)); //adding the information
to the vector
    }
    return partitionList;
```

```
}

void FirstFit(vector<Job> jobList,
vector<Partition> partitionList) //
first fit algorithm
{
    int totalWaste = 0;
    for (int i = 0; i <
jobList.size(); i++) // going through
the vector of job objects
    {
        for (int j = 0; j <
partitionList.size(); j++) //going
through the vector of partition
objects
        {
            if ((jobList[i].getSize()
<= partitionList[j].getSize()) && (!
jobList[i].getStatus() && !
partitionList[j].getAssigned()))
```

```
                {

jobList[i].setStatus(true);  //
assigning the process status to true

partitionList[j].setWaste(partitionLis
t[j].getSize() -
jobList[i].getSize());

jobList[i].setWaste(partitionList[j].g
etWaste()); //setting the waste of the
process to the waste of the partition

jobList[i].setPartitionNum(partitionLi
st[j].getPartition_id());

partitionList[j].setAssign(true); //
assigning the partition to in use
```

```
partitionList[j].setprocess_ID(jobList
[i].getProcess_id());
                totalWaste +=
partitionList[j].getWaste(); //getting
the value for totalwaste by getting
the waste from the partition
                break;
            }
        }
    }
    cout <<
"----------------------------------
-----------------";
    cout << "\n\n\t\t"
         << "First Fit Algorithm" <<
endl;
    print(jobList, partitionList,
totalWaste); //calling print function
to output data
```

```
}

void NextFit(vector<Job> jobList,
vector<Partition> partitionList) //
nextFit algorithm
{
    int j = 0,
            k = 0;
    int totalWaste = 0;
    for (int i = 0; i <
jobList.size(); i++) // going through
the vector of job objects
    {
        while (k <
partitionList.size()) //going through
the vector of partition objects using
a while loop
        {
```

```
            if ((jobList[i].getSize()
<= partitionList[j].getSize()) && (!
jobList[i].getStatus() && !
partitionList[j].getAssigned()))
            {

jobList[i].setStatus(true);

jobList[i].setPartitionNum(partitionLi
st[j].getPartition_id()); //sassigning
a partition to a job by getting the
partition by its id

partitionList[j].setWaste(partitionLis
t[j].getSize() -
jobList[i].getSize());

jobList[i].setWaste(partitionList[j].g
etWaste()); //setting the waste of the
process by getting the waste that was
from the partition
```

```
partitionList[j].setAssign(true);

partitionList[j].setprocess_ID(jobList
[i].getProcess_id()); //setting the
process id of a partition by getting a
process by its id
                totalWaste +=
partitionList[j].getWaste();
                break;
            }
            k++;
            j = ((j + 1) %
partitionList.size());
        }
    }
    cout <<
"----------------------------------
-----------------";
```

```
    cout << "\n\n\t\t"
         << "Next Fit Algorithm" <<
endl;
    print(jobList, partitionList,
totalWaste); //calling print function
to output data
}

void BestFit(vector<Job> jobList,
vector<Partition> partitionList) //
BestFit algorithm
{
    int totalWaste = 0;
    int Best_id = -1;
    for (int i = 0; i <
jobList.size(); i++) // going through
the vector of job objects
    {
        Best_id = -1;
```

```
        for (int j = 0; j <
partitionList.size(); j++) //going
through the vector of partition
objects
        {
            if ((jobList[i].getSize()
<= partitionList[j].getSize()) && (!
jobList[i].getStatus() && !
partitionList[j].getAssigned()))
            {
                if (Best_id == -1)
                {
                    Best_id = j;
                }
                else if
(partitionList[Best_id].getSize() >
partitionList[j].getSize())
                {
                    Best_id = j;
```

```
                    }
                }
            }
            if (Best_id != -1)
            {

jobList[i].setStatus(true); //setting
the status of a job to true

jobList[i].setPartitionNum(partitionLi
st[Best_id].getPartition_id()); //
assigning the process to a partition

partitionList[Best_id].setWaste(partit
ionList[Best_id].getSize() -
jobList[i].getSize()); //calculating
the waste of the partition and setting
the waste
```

```
jobList[i].setWaste(partitionList[Best
_id].getWaste());

partitionList[Best_id].setAssign(true)
;

partitionList[Best_id].setprocess_ID(j
obList[i].getProcess_id()); //setting
the process id of the partition
                totalWaste +=
partitionList[Best_id].getWaste(); //
updating the totalWaste value

            }
        }
        cout <<
"-----------------------------------
------------------";
        cout << "\n\n\t\t"
```

```
        << "Best Fit Algorithm" <<
endl;
    print(jobList, partitionList,
totalWaste); //calling print function
to output data
}

void WorstFitFixed(vector<Job>
jobList, vector<Partition>
partitionList) //WorstFitFixed
algorithm
{
    int totalWaste = 0;
    int worst_id = -1;
    for (int i = 0; i <
jobList.size(); i++) // going through
the vector of job objects
    {
        worst_id = -1;
```

```
        for (int j = 0; j <
partitionList.size(); j++)  //going
through the vector of partition
objects
        {
            if ((jobList[i].getSize()
<= partitionList[j].getSize()) && (!
jobList[i].getStatus() && !
partitionList[j].getAssigned()))
            {
                if (worst_id == -1)
                {
                    worst_id = j;
                }
                else if
(partitionList[worst_id].getSize() <
partitionList[j].getSize())
                {
                    worst_id = j;
```

```
                    }
                }
            }
            if (worst_id != -1)
            {

jobList[i].setStatus(true); //setting
the process to the value of true for
its status

jobList[i].setPartitionNum(partitionLi
st[worst_id].getPartition_id()); //
setting a partition number to a
process

partitionList[worst_id].setWaste(parti
tionList[worst_id].getSize() -
jobList[i].getSize());

jobList[i].setWaste(partitionList[wors
```

```
t_id].getWaste()); //setting the waste
of a process by seeing the waste from
the partition

partitionList[worst_id].setAssign(true
);

partitionList[worst_id].setprocess_ID(
jobList[i].getProcess_id());
                totalWaste +=
partitionList[worst_id].getWaste(); //
updating the totalwaste
            }
        }
        cout <<
"----------------------------------
------------------";
        cout << "\n\n\t\t"
             << "Worst Fit (Fixed)
Algorithm" << endl;
```

```
    print(jobList, partitionList,
totalWaste); //calling print function
to output data
}

void WorstFitDynamic(vector<Job>
jobList, vector<Partition>
partitionList) //WorstFitDynamic
algorithm
{
    int totalWaste = 0;
    int worst_id = -1;
    for (int i = 0; i <
jobList.size(); i++) // going through
the vector of job objects
    {
        worst_id = -1;
        for (int j = 0; j <
partitionList.size(); j++)  //going
```

```
through the vector of partition
objects
        {
            if ((jobList[i].getSize()
<= partitionList[j].getSize()) && (!
jobList[i].getStatus() && !
partitionList[j].getAssigned()))
            {
                if (worst_id == -1)
                {
                    worst_id = j;
                }
                else if
(partitionList[worst_id].getSize() <
partitionList[j].getSize())
                {
                    worst_id = j;
                }
            }
```

```
        }
        if (worst_id != -1)
        {

jobList[i].setStatus(true); //setting
the value of the process to true

jobList[i].setPartitionNum(partitionLi
st[worst_id].getPartition_id()); //
assigning the process to a partition

partitionList[worst_id].setWaste(parti
tionList[worst_id].getSize() -
jobList[i].getSize());

jobList[i].setWaste(partitionList[wors
t_id].getWaste()); //setting the waste
of the process by getting the waste
left from the partition
```

```
partitionList[worst_id].setAssign(true
);

partitionList[worst_id].setprocess_ID(
jobList[i].getProcess_id());

            if
(partitionList[worst_id].getWaste() >
0) //condition that checks if the
waste for a partition is greater than
zero
            {

partitionList.insert(partitionList.beg
in() + worst_id + 1,
Partition(partitionList[worst_id].getP
artition_id() + 0.1,
partitionList[worst_id].getWaste()));
```

```
partitionList[worst_id].setSize(partit
ionList[worst_id].getSize() -
partitionList[worst_id].getWaste()); /
/setting the size of a partition with
the worst_id

partitionList[worst_id].setWaste(0); /
/setting the waste of the partition
with the worst id to zero
            }
            totalWaste +=
partitionList[worst_id].getWaste();
        }
    }
    //calling print function to output
data
    cout <<
"----------------------------------
-----------------";
```

```
    cout << "\n\n\t\t"
         << "Worst Fit (DYNAMIC)
Algorithm" << endl;
    print(jobList, partitionList,
totalWaste);
    cout << "\n\t ------- Main Memory
Configuration -------\n";
    cout << "Partitions\t\t"
         << "Partition Size\t\t" <<
"Job Assigned" << endl;

    for (int i = 0; i <
partitionList.size(); i++)
    {

if(partitionList[i].getprocess_ID() ==
-1) {
            cout
<<partitionList[i].getPartition_id()
```

```
<< "\t\t\t" <<
partitionList[i].getSize() << "\t\t\t"
<< "-" << endl;;
        } else {
            cout <<
partitionList[i].getPartition_id() <<
"\t\t\t" << partitionList[i].getSize()
<< "\t\t\t" <<
partitionList[i].getprocess_ID() <<
endl;
        }
    }
}
```