OS PROJECT REVIEW

TOPIC: SMART TRAVEL MANAGEMENT

SUBMITTED TO: PROF. Amrit Pal

SUBMITTED BY:

Harsh Sharma 19BCE1464
Shreeharsh Pandey 19BCE1818

SLOT:L33+L34

COURSE CODE: CSE2005

SMART TRAVEL MANAGEMENT

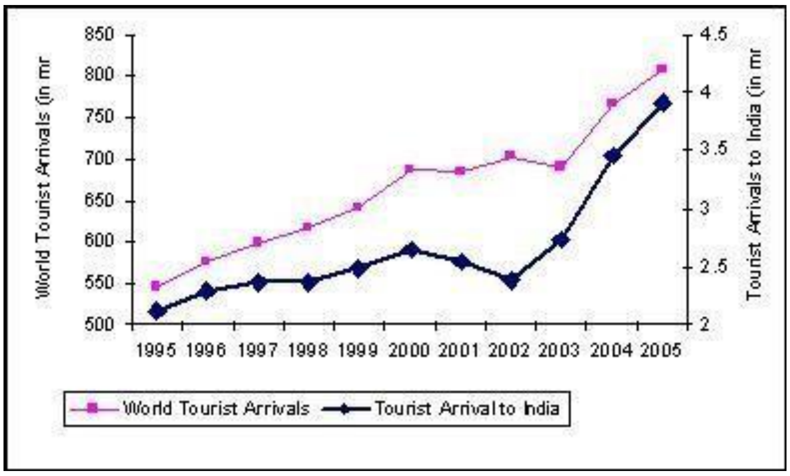
Abstract

Tourism in India generated around 6.3% of the nation's GDP in the year 2015 alongside supporting 37.315 million occupations, which makes for around 8.7% of the aggregate business. Starting at 2016, the eventual fate of tourism in India looks splendid and prosperous as a result of the different positive perspectives in the nation. With awesome visitor spots, various geography, fluctuated societies and legacy destinations in the nation, India is a standout amongst the most lucrative choices for sightseers. The tourism business of the nation has experienced different changes over the time. Due to heavy income associated with Tourism, it is highly essential to carry out this aspect in a profitable way. Travel agents represent the primary intermediary of travel-related products. They account for 40% of the domestic and 60% of the international airline ticket sales; almost all the cruise and tour package sales and during the period 2006/2007, they sold 30 million room nights. So it is the job of these travel agents so as to book the trips in an efficient as well as a profitable way.

Keywords : Algorithm, scheduling, waiting time, start-date, closing date, priority, random.

Introduction

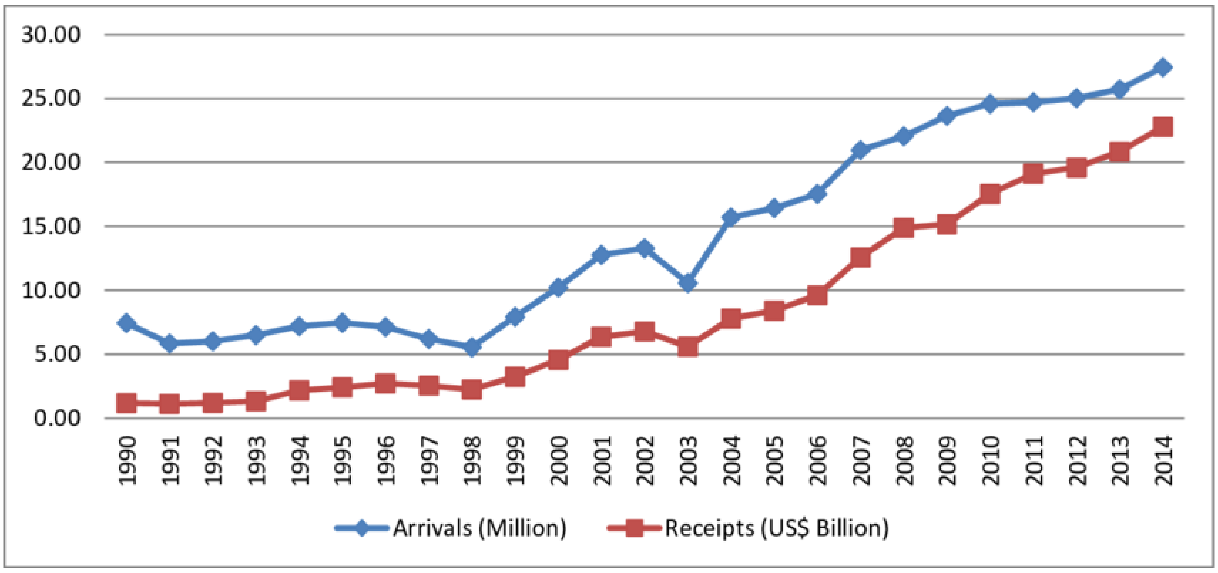
Throughout the years, tourism has kept on increasing huge enthusiasm at a worldwide scale. It is a noteworthy outside trade worker for a decent number of cutting edge and developing economies. It is likewise obvious that data blast makes it unwieldy occasions to get to pertinent data to upgrade basic leadership. This has offered ascend to the rise of wise frameworks or systems that encourage snappy access to pertinent substance found in the Internet.



Since the turn of a millennium, the Travel and Tourism industry has seen changes of radical extents, most quite the methods by which holidaymakers are sorting out and booking their

ideal breaks, alongside the levels of rivalry that presently exists inside this tenacious industry.

Client benefit has in this manner turn into an unequivocal factor while selecting travel organizations. This expanding client desire for travel, has implied to operators that various new key difficulties must be tended to while going ahead. The techniques in which fly out operators battle these difficulties will at last figure out who will flourish or scarcely survive. One of the difficulties they face is scheduling the trips in an efficient and smooth way. Planning a trip is an arduous task for both the operator and the customer. Due to theever-increasing number of tourists, operator should be imbibe newer methods to cope with the traffic.(KPS, n.d.)



Regardless of the prospering offer of the Asian market, India's aggregate offer in world traveler landings, be that as it may, remains an unassuming 0.8%, despite the fact that global entries to India have seen to some degree a sensational turnaround since 2002, when an impermanent declining pattern was switched forcefully. This turnaround was the aftereffect of a few factors, for example, Government of India's "Incredible India" crusade, high perceivability stood to India by its financial achievement, the tourism business' consistent look for new goals, and to some degree change in framework in particular regions, (for example, better air network of littler and remote goals).

This above data very much tells us that if used in an efficient way, India can be a huge generator of tourism revenue. For this to achieve there has a radical change in the way we deal with Tourism. This can include using a new Traveler Monitoring system which would monitor the way tourists perceive the service and act.

At present, the development of China's tourism data framework is obsolete for long haul advancement later on. 2010 Shanghai World Expo invigorated the advancement of tourism, and individuals' material expectation for everyday comforts likewise has been extraordinarily enhanced, correspondingly household and abroad tourism has been created quickly. Insights by the National Bureau of Statistics demonstrated that: in 2012, 2013, 2014

the quantity of household voyagers separately achieved 1.802 billion, 1.923 billion and 2.212 billion.

Literary Review

__**Tourism Knapsack Optimization for Generic Algorithm**__(Shi, n.d.)

In this proposed tourism management system, there is a problem needed to be solved: in the planned playtime, visitors can select sights with praise as high as possible. This can be attributed to the following knapsack problem: backpack capacity T defined as planned play time, P as the rating, sights are represented with 1,2, ,,, N as the playtime of sight Li denoted Li, indicates rate of sight i=1,2,3,4,,,n . In additional, use x to indicate whether select sights or not.

Then the above knapsack problem is transformed into the following optimization problem:

$$\begin{pmatrix} r \\ s \\ t \end{pmatrix} = \begin{bmatrix} y_{10} & y_{11} & y_{12} \\ y_{20} & y_{21} & y_{22} \\ y_{30} & y_{31} & y_{32} \end{bmatrix}$$

In the genetic algorithm, the chromosome is composed as binary string with 0 and 1.

In the genetic algorithm, the randomly generated initial population, as well as cross-new chromosomal aberrations may not meet the constraints of problem.(Shiyu Wang, n.d.)

$$T = \begin{bmatrix} 1 & o & o \\ o & c_x & s_y \\ o & -y_v & c_x \end{bmatrix} \times \begin{bmatrix} c_y & o & s_v \\ o & 1 & o \\ -s_v & o & c_x \end{bmatrix} \times \begin{bmatrix} c_v & s_v & o \\ -s_v & c_v & o \\ o & o & 2 \end{bmatrix}$$

Therefore, we use the following greedy method for chromosome amending.

Step1. For chromosome $i - x, y, z$, its corresponding sights are c, c, c, then put them in descending order of the highest praise to form a new attraction serial number.

Step2. k=1;

Step3. If $p(\xi|\phi) - \Sigma_{k-1}^{n} w_i t_i(\zeta)$, them turn to Step3， otherwise turn to Step4.

Step4. Suppose that (x, y, z, t)

Step5. Set all genes of chromosome $i - x, y, z$ to 0 which are corresponding with c(i), $\Sigma_{k-1}^{n} w_i - 1, t_i(\zeta)$,to form a new chromosome $G_{aaz} - s_{iaa}$,so in this case, Σ is a chromosome that meets condition (3-2).

<u>Travelling Salesman Problem</u>(Chow, n.d.)

The traveling salesman problem consists of a salesman and a set of cities. The salesman has to visit each one of the cities starting from a certain one (e.g. the hometown) and returning to the same city. The challenge of the problem is that the traveling salesman wants to minimize the total length of the trip.

The traveling salesman problem can be described as follows:

TSP = {(G, f, t): G = (V, E) a complete graph,

f is a function V×V Z, → t ∈ Z,

G is a graph that contains a traveling salesman tour with cost that does not exceed t}.

The traveling salesman problem is NP-complete.

As per Karp, we can parcel the issue to get an inexact arrangement utilizing the gap and overcome systems. We shape gatherings of the urban communities and find ideal visits inside these gatherings. At that point we join the gatherings to locate the ideal voyage through the first issue. Karp has given probabilistic examinations of the execution of the calculations to decide the normal mistake and accordingly the normal execution of the arrangement contrasted with the ideal arrangement. Karp has proposed two partitioning plans. As indicated by the principal, the urban areas are isolated as far as their area and as it were. As per the second, the urban areas are isolated into cells that have a similar size. Karp has given upper limits of the most pessimistic scenario mistake for the principal partitioning plan, or, in other words Adaptive Dissection Method. The working of this strategy is clarified underneath. (Y.A. Adekunle1, Z.O. Ogunwobi2, A. Sarumi Jerry3, B.T. Efuwape2, Seun Ebiesuwa1, n.d.)

Let us assume that the n urban areas are circulated in a square shape. This square shape is separated in B = 2k sub square shapes. Each sub square shape contains at most t urban areas where k = log2[(N-1)/(t-1)]. The calculation processes an ideal visit for the urban communities inside a sub-square shape. These 2k ideal visits are consolidated to locate an ideal visit for the N urban areas. Give us a chance to clarify the working of the division calculation. Expect Y to be a square shape with num being the quantity of urban areas. The square shape is separated into two square shapes in correspondence of the [num/2] th city from the shorter side of the square shape. This city is valid that has a place with the regular limit of the two square shapes. Whatever is left of the division procedure is done recursively. The outcomes for this calculation are introduced beneath. Accept a square shape X containing N urban areas and t the greatest permitted number of urban areas in a sub-square shape. Accept W to be the length of the walk the calculation gives and Lopt to be the length of the ideal way.

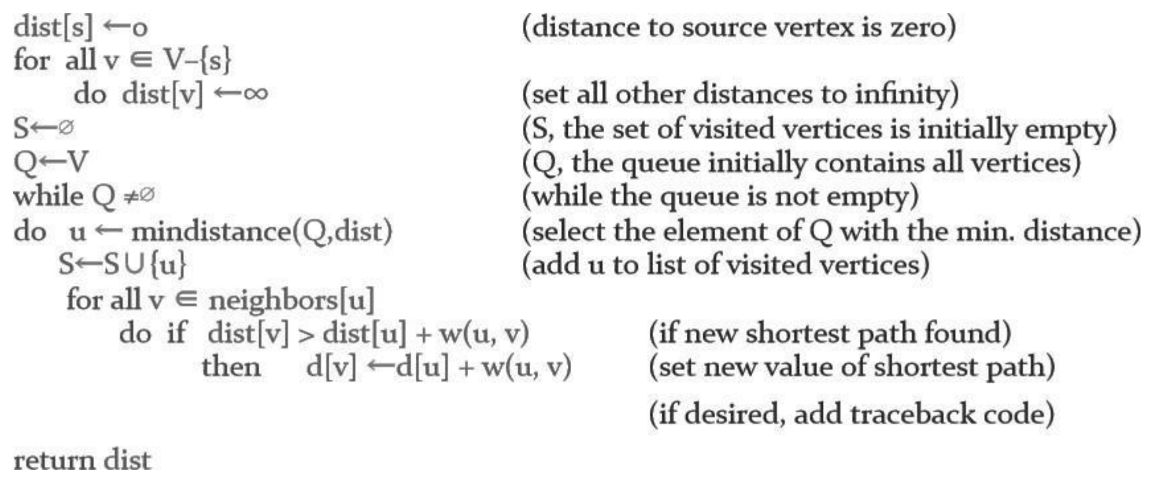
The focuses circulation does not influence the outcome. It ought to be noted anyway that these outcomes hold for uniform conveyances. We currently accept irregular disseminations to sum up the outcomes.

Dijkstra Algorithm(ChitraNayal, n.d.)

It is an Algorithm for finding the shortest paths between nodes in a graph, which may represent, for examples, travel schedules in my case, road network.

It was first conceived by Dutch computer scientist Edsger W. Dijkstra in 1956 and published three years later.

Dijkstra's method is fundamentally the same as Prim's method for least crossing tree. Like Prim's MST, we produce a SPT (briefest way tree) with given source as root. We keep up two sets, one set contains vertices incorporated into most brief way tree, other set incorporates vertices not yet incorporated into briefest way tree. At each progression of the calculation, we discover a vertex which is in the other set (arrangement of not yet included) and has a base separation from the source.



```
dist[s] ←o                             (distance to source vertex is zero)
for  all v ∈ V–{s}
      do  dist[v] ←∞                   (set all other distances to infinity)
S←∅                                    (S, the set of visited vertices is initially empty)
Q←V                                    (Q, the queue initially contains all vertices)
while Q ≠∅                             (while the queue is not empty)
do   u ← mindistance(Q,dist)           (select the element of Q with the min. distance)
     S←S∪{u}                           (add u to list of visited vertices)
      for all v ∈ neighbors[u]
            do  if   dist[v] > dist[u] + w(u, v)        (if new shortest path found)
                    then     d[v] ←d[u] + w(u, v)       (set new value of shortest path)
                                                        (if desired, add traceback code)
return dist
```

Creating planning calculations and understanding their effect practically speaking have been of enthusiasm since the mid 1970s. Scientists have demonstrated that, consolidating in excess of one booking strategy enhances execution .

In their paper , Neetu(Yan, n.d.) et al. exhibited a state chart that delineates the near investigation of different booking calculations for a solitary CPU and demonstrated which calculation is best for the specific circumstance. Their paper gave a portrayal on what is happening inside the framework and why an alternate arrangement of procedures is a possibility for the distribution of the CPU at various occasions. Basically, the goal of their paper was to dissect the high productive CPU scheduler on outline of the top notch planning calculations which suits the booking objective. In any case, they were not ready to touch base at an unequivocal end. Other than this work, Edwin et al. in concentrated the booking of errands in PC frameworks which use loose (halfway) calculations. In their framework, undertakings arrive haphazardly amid run-time. Each errand has two levels of calculation time necessity: the full level calculation prerequisite and the decreased level calculation necessity. They at long last dedicated their consideration regarding build up an express recipe for the for all intents and purposes vital execution measurements, for example, the standardized mean undertaking holding up time, the mean errand served calculation time, and the division of assignments completely handled. Another paper firmly identified with our paper is. In their paper, Ankur et al. made an examination of existing planning calculation based on seven parameters specifically acquisition, many-sided quality, allotment, application, holding up time, ease of use, and sort of framework.

They additionally saw that there is nobody calculation fulfilling all the seven premise parameters decided for investigation and recommended requirements for development. Be that as it may, no heading for the change was given and has been alluded to as future works. Notwithstanding these related papers, K.M. et al., in overviewed inquire about on planning calculations, checked on past arrangements of booking issues and displayed an order conspire. Utilizing a uniform phrasing for booking techniques and the new characterization plot, they distinguished patterns in planning research lastly finished up their work by introducing a strategy for creating booking procedures.

This and others papers [8, 9, 10, 11, 12], however not identified with our paper, exhibited planning calculations utilized in other processing fields.

Scheduling Criteria

Waiting Time:

This is the total amount of time that a process is in the ready queue waiting in order to be executed. Sometimes the waiting time could be short and at other times it could be long.

FIRST COME FIRST SERVED (FCFS)

First-Come-First-Served calculation is known as the least complex booking calculation. Here, forms are dispatched by their landing time on the prepared line. Named non-preemptive booking calculation, it races to culmination once admitted to the CPU. In this manner, it alludes to as « keep running until done » and utilizations FIFO (First-In-First-Out) calculation to complete the assignment. In other term, it implies that one program runs non-preemptively until the point when it is done (counting any hindering for I/O activity). That is, keep the CPU until done. Along these lines, the FCFS plot isn't helpful in booking intelligent clients since it can't ensure great reaction time. Easy to compose and comprehend; its significant downside is that the normal time is frequently very long. Additionally, the FCFS is uncalled for as in long employments make short occupations pause and irrelevant long occupations make critical short employments pause. In any case, it is more unsurprising than the greater part of alternate plans since it offers time. Truly, the FCFS is infrequently utilized as a plan in current working frameworks yet usually installed inside different plans.

SHORTEST JOB FIRST (SJF)

Shortest Job First (SJF) is a booking calculation that chooses the holding up process with the littlest execution time to execute straightaway. Another name for SJF is Shortest Process Next Algorithm. Briefest occupation initially is worthwhile due to its straightforwardness and in light of the fact that it amplifies process throughput (as far as the quantity of procedures raced to finishing in a given measure of time). In any case, it has the potential for process starvation for procedures which will require quite a while to finish if short procedures are included consistently. Most astounding reaction proportion next is comparative yet gives an answer for this issue. Briefest occupation next booking is once in a while utilized outside of particular conditions since it requires precise estimations of the runtime of all procedures that are sitting tight for execution. It is provably ideal as respects the minimization of the normal holding up time. It works for both preemptive and non-preemptive schedulers.

RANDOM SCHEDULING

Random scheduling is a probabilistic scheduling algorithm for processes in an operating system. Processes are each assigned some number of lottery tickets, and the scheduler draws a random ticket to select the next process to be executed. The distribution of tickets need not be uniform; granting a process more tickets provides it a relative higher chance of selection. This technique can be used to approximate other scheduling algorithms, such as shortest job next and Fair-share scheduling. Lottery scheduling solves the problem of starvation. Giving each process at least one lottery ticket guarantees that it has non-zero probability of being selected at each scheduling operation. On average, CPU time is proportional to the number of tickets given to each job. For approximation in SJF, most tickets are assigned to short running jobs, and fewer to longer running jobs. To avoid starvation, every job gets at least one ticket. This is highly inefficient if the number of the trips or objects to be allotted numbers increases.

PRIORITY SCHEDULING

This one is the same as the random scheduling but in this case the lottery number or the priority are assigned based on the requirement of the Customer when booking his/her trip.

Our Work

In our undertaking study we attempt to make a database of individuals who want to visit their places of enthusiasm with the assistance of this organization. We will make a customized client access for every one of the clients which would include the storage of their personal details like their name and other details along with their preferred place to travel to and their preferred time to travel.

We endeavor to plan their trips in a way the sitting tight time for them to visit a place turns out to be less when contrasted with holding up quite a while to get into a trip group. We will present another alternative called as need for which they need to pay a touch of additional cash with the goal that their excursion gets planned at an earlier time as contrasted to the one which is free.

Proposed Method

We plan to compare 4 different algorithms and compare their pros and cons.

The first algorithm we want to use is called First Come First serve. In this we schedule the given trips based on the date on which they were booked. The process for it is simple. Sort the trips according to their booking dates and schedule them accordingly.

The second is called the shortest job first algorithm which schedules the trips based on their trip time. This is useful if there is a lot of variation in the trip times.

The third is called the Priority algorithm. Their trips would be planned in light of the needs they picked while they were reserving their trips.

In the random way of scheduling their trips would be planned in a random order without considering any other factors. We endeavor to plan their time of travel in light of the above calculations and contrast their holding up or sitting times inclining toward lesser ones and their favored time of travel is accomplished.

Code

```
#include<iostream>
#include<stdio.h>
#include<bits/stdc++.h>
#include<string.h>
#define DAY 1
#define MONTH 1
#define YEAR 2018

using namespace std;

struct waiting_times
{
    char s[50];
    float wt;
```

```
}fwt[4];

struct process
{
int at,bt,pr,pno;
};

struct Date
{
   int d, m, y;
};

const int monthDays[12] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

int countLeapYears(Date d)
{
   int years = d.y;
   if (d.m <= 2)
      years--;
   return years / 4 - years / 100 + years / 400;

}

bool isLeap(int y)
{
   if (y%100 != 0 && y%4 == 0 || y %400 == 0)
      return true;
   return false;
}

int offsetDays(int d, int m, int y)
{
   int offset = d;

   switch (m - 1)
   {
   case 11:
      offset += 30;
   case 10:
      offset += 31;
   case 9:
      offset += 30;
   case 8:
      offset += 31;
   case 7:
      offset += 31;
   case 6:
      offset += 30;
   case 5:
      offset += 31;
   case 4:
      offset += 30;
   case 3:
      offset += 31;
   case 2:
      offset += 28;
   case 1:
      offset += 31;
```

```
    }

    if (isLeap(y) && m > 2)
        offset += 1;

    return offset;
}

void revoffsetDays(int offset, int y, int *d, int *m)
{
  int month[13] = { 0, 31, 59, 90, 120, 151, 181,
                    212, 243, 273, 304, 334, 365 };

    if (isLeap(y))
        month[2] = 29;

    int i;
    for (i = 1; i <= 12; i++)
    {
        if (offset <= month[i])
            break;
        offset = offset - month[i];
    }

    *d = offset;
    *m = i;
}

int getDifference(Date dt1, Date dt2)
{
    long int n1 = dt1.y*365 + dt1.d;
    for (int i=0; i<dt1.m - 1; i++)
        n1 += monthDays[i];
    n1 += countLeapYears(dt1);
    long int n2 = dt2.y*365 + dt2.d;
    for (int i=0; i<dt2.m - 1; i++)
        n2 += monthDays[i];
    n2 += countLeapYears(dt2);
    return (n2 - n1);
}

int date_days(Date dt2)
{
    Date dt1={1, 1, 2018};
    return getDifference(dt1, dt2);
}

void days_date(int x)
{
    int offset1 = offsetDays(DAY, MONTH, YEAR);
    int remDays = isLeap(YEAR)?(366-offset1):(365-offset1);
    int y2, offset2=offset1;
    if (x <= remDays)
    {
        y2 = YEAR;
        offset2 += x;
    }
    else
```

```
   {
      x -= remDays;
      y2 = YEAR + 1;
      int y2days = isLeap(y2)?366:365;
      while (x >= y2days)
      {
         x -= y2days;
         y2++;
         y2days = isLeap(y2)?366:365;
      }
      offset2 = x;
   }
   int m2, d2;
   revoffsetDays(offset2, y2, &d2, &m2);

   cout <<d2<<":"<<m2<<":"<<y2;
}

//FCFS

void F_findWaitingTime(int processes[], int n, int bt[], int wt[], int at[])
{
   int service_time[n];
   service_time[0] = 0;
   wt[0] = 0;
   for (int i = 1; i < n ; i++)
   {
      service_time[i] = service_time[i-1] + bt[i-1];
      wt[i] = service_time[i] - at[i];
       if (wt[i] < 0)
          wt[i] = 0;
   }
}
void F_findTurnAroundTime(int processes[], int n, int bt[], int wt[],
                  int tat[])
{
   for (int i = 0; i < n ; i++)
      tat[i] = bt[i] + wt[i];
}
void F_findavgTime(int processes[], int n, int bt[], int at[])
{
   int wt[n], tat[n], compl_time[n];

   F_findWaitingTime(processes, n, bt, wt, at);

   // Display processes along with all details
   cout << "Trip " << " Trip Time " << " Start Date "<<
    " Completion Date " << " Waiting Time \n";
   int total_wt = 0;
   for (int i = 0 ; i < n ; i++)
   {
      total_wt = total_wt + wt[i];
      cout << " " << i+1 << " \t " << bt[i]<<"\t ";
      days_date(at[i]+wt[i]);
      cout<<" \t ";
      days_date(wt[i]+bt[i]+at[i]);
      cout<<"\t " << wt[i] << "\t ";
      cout<< endl;
   }

   cout << "\nAverage waiting time = "
       << (float)total_wt / (float)n;
   strcpy(fwt[0].s,"First Come First Serve");
   fwt[0].wt= (float)total_wt / (float)n;
   cout<<"\n\n";
}


//SJF

void Snew_findavgTime(int at[], int bt[], int n)
{
   int i,p[10]={1,2,3,4,5,6,7,8,9,10},min,k=1,btime=0;
   int temp,j,wt[10],tt[10],ta=0,sum=0;
   float wavg=0,tavg=0,tsum=0,wsum=0;
   for(i=0;i<n;i++)
   {
      for(j=0;j<n;j++)
      {
          if(at[i]<at[j])
          {
              temp=p[j];
              p[j]=p[i];
              p[i]=temp;
              temp=at[j];
              at[j]=at[i];
              at[i]=temp;
              temp=bt[j];
              bt[j]=bt[i];
              bt[i]=temp;
          }
      }
   }

 for(j=0;j<n;j++)
 {
 btime=btime+bt[j];
 min=bt[k];
 for(i=k;i<n;i++)
 {
 if (btime>=at[i] && bt[i]<min)
 {
 temp=p[k];
 p[k]=p[i];
 p[i]=temp;
 temp=at[k];
 at[k]=at[i];
 at[i]=temp;
 temp=bt[k];
 bt[k]=bt[i];
 bt[i]=temp;
 }
 }
 k++;
 }
 wt[0]=0;
 for(i=1;i<n;i++)
 {
 sum=sum+bt[i-1];
 wt[i]=sum-at[i];
 wsum=wsum+wt[i];
 }

 wavg=(wsum/n);
 for(i=0;i<n;i++)
 {
 ta=ta+bt[i];
 tt[i]=ta-at[i];
 tsum=tsum+tt[i];
 }

 tavg=(tsum/n);
 cout << "Trip " << " Trip Time " << " Start Date "<<
  " Completion Date " << " Waiting Time \n";
  for (int i = 0 ; i < n ; i++)
  {
     cout << " " << p[i] << " \t " << bt[i] << "\t ";
     days_date(at[i]+wt[i]);
     cout<<" \t ";
     days_date(bt[i]+wt[i]);
     cout<<"\t " << wt[i] << "\t ";
     cout<< endl;
  }

  cout << "\nAverage waiting time = "<< wavg <<endl;
  strcpy(fwt[1].s,"Shortest Job First");
  fwt[1].wt=wavg;
  cout<<"\n\n";
}

//Priority

bool comp(process a,process b)
{
if(a.at == b.at)
{
return a.pr<b.pr;
}
else
{
  return a.at<b.at;
}
}

// Using FCFS Algorithm to find Waiting time
void get_wt_time(process proc[], int wt[], int totalprocess)
{
  int service[50];
  service[0]=0;
  wt[0]=0;
  for(int i=1;i<totalprocess;i++)
  {
     service[i]=proc[i-1].bt+service[i-1];
     wt[i]=service[i]-proc[i].at+1;
```

```
            if(wt[i]<0)
            {
            wt[i]=0;
            }
    }
}

void get_tat_time(process proc[], int tat[],int wt[], int totalprocess)
{
for(int i=0;i<totalprocess;i++)
{
    tat[i]=proc[i].bt+wt[i];
}
}

void P_findavgTime(process proc[], int totalprocess, int choice)
{
//Declare waiting time and turnaround time array
int wt[50],tat[50];
double wavg=0,tavg=0;

get_wt_time(proc, wt, totalprocess);
get_tat_time(proc, tat,wt, totalprocess);

int stime[50],ctime[50];
stime[0]=0;
ctime[0]=stime[0]+tat[0];
// calculating starting and ending time
for(int i=1;i<totalprocess;i++)
    {
        stime[i]=ctime[i-1];
        ctime[i]=proc[i].at+proc[i].bt+wt[i];
    }

cout << "Trip " << " Trip Time " <<"Priority"<< " Start Date "<<
     " Completion Date " << " Waiting Time \n";
for(int i=0;i<totalprocess;i++)
    {
        wavg += wt[i];
        cout<<" "<<proc[i].pno<<"    \t "<<proc[i].bt<<"    \t     "<<
            proc[i].pr<<"      ";
        days_date(stime[i]);
        cout<<"\t ";
        days_date(ctime[i]);
        cout<<"\t "<<wt[i]<<"\n"<<endl;
    }

    cout<<"Average waiting time is : ";
    cout<<wavg/(float)totalprocess<<endl;
    if(!choice)
      {
        strcpy(fwt[2].s,"Priority Scheduling");
        fwt[2].wt=wavg/(float)totalprocess;
      }
    else
    {
      strcpy(fwt[3].s,"Random Scheduling");
      fwt[3].wt=wavg/(float)totalprocess;
```

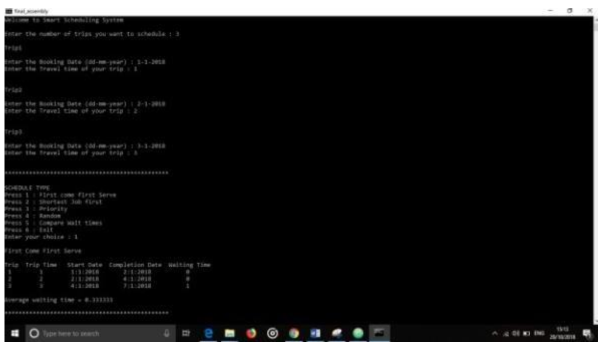
```
  }
 cout<<"\t\t"<<endl;
}

int main()
{
   srand(time(0));
   int number;
   printf("Welcome to Smart Scheduling System\n\n");
   printf("Enter the number of trips you want to schedule : ");
   scanf("%d", &number );
   cout<<"\n";
   int choice, m, i, tr_time[number], processes[number],
      burst_time[number];
   Date dt[number];
   for(m=0; m<number; m++)
   {
      cout<<"Trip"<<m+1<<"\n"<<endl;
      printf("Enter the Booking Date (dd-mm-year) : ");
      scanf("%d-%d-%d", &dt[m].d, &dt[m].m, &dt[m].y);
      cout<<"Enter the Travel time of your trip : ";
      cin>>tr_time[m];
      cout<<"\n\n";
   }
   do
   {
      cout<<"***********************************************"<<endl;
      cout<<"\nSCHEDULE TYPE"<<endl;
      cout<<"Press 1 : First come first Serve"<<endl;
      cout<<"Press 2 : Shortest Job first"<<endl;
      cout<<"Press 3 :  Priority"<<endl;
      cout<<"Press 4 : Random"<<endl;
      cout<<"Press 5 : Compare Wait times"<<endl;
      cout<<"Press 6 : Exit "<<endl;
      cout<<"Enter your choice : ";
      cin>>choice;
      cout<<"\n";
      switch(choice)
      {
         case 1://FCFS
         {
            cout<< "First Come First Serve\n"<<endl;
            int farrival_time[number];
            for(i=0; i<number; i++)
            {
               processes[i]=i;
               burst_time[i]=tr_time[i];
               farrival_time[i]=date_days(dt[i]);
            }
            F_findavgTime(processes, number, burst_time, farrival_time);
            break;
         }

         case 2://SJF
         {
            cout<<" Shortest Job First\n\n"<<endl;
            int snbt[number], snat[number];
            for(i=0; i<number; i++)
```

```
        {
           snat[i]=date_days(dt[i]);
           snbt[i]=tr_time[i];
        }
        Snew_findavgTime(snat, snbt, number);
        break;
     }
     case 3://Priority
     {
        cout<<"Priority Scheduling\n"<<endl;
        process proc[number];
        for(i=0; i<number; i++)
        {
           proc[i].at=date_days(dt[i]);
           proc[i].bt=tr_time[i];
           cout<<"Enter the Priority of Process "<<i+1<<" : ";
           cin>>proc[i].pr;
           proc[i].pno=i+1;
        }
        sort(proc,proc+number,comp);
        P_findavgTime(proc, number,0);
        break;
     }
     case 4://Random Scheduling
     {
       cout<<"Random Scheduling\n"<<endl;
       process pproc[number];
       int ppriority[number];
       for(i=0; i<number; i++)
          ppriority[i]=i+1;
       for(i=0; i<number; i++)
       {
          pproc[i].at=date_days(dt[i]);
          pproc[i].bt=tr_time[i];
          cout<<"Random entry of Priorities\n\n ";
          pproc[i].pr=(ppriority)[rand() % number];
          pproc[i].pno=i+1;
       }
       sort(pproc,pproc+ number,comp);
       P_findavgTime(pproc,number,1);
       break;
     }
     case 5:
     {
        cout<<"\tAlgorithm\t\t"<<"Waiting Time\n";
        for(i=0;i<4;i++)
        {
           cout<<fwt[i].s<<"\t ";
           cout<<fwt[i].wt;
           cout<<"\n";
        }
        break;
     }

   case 6:
       exit(1);
  }
```

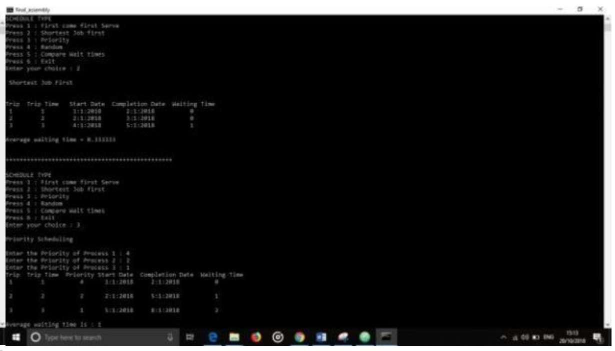
```
    }while(choice < 5);
 }
```

Results

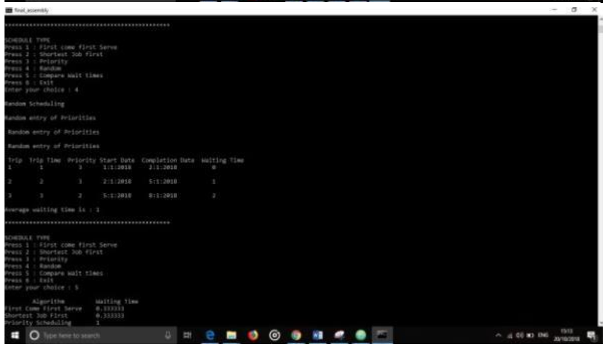
SET 1



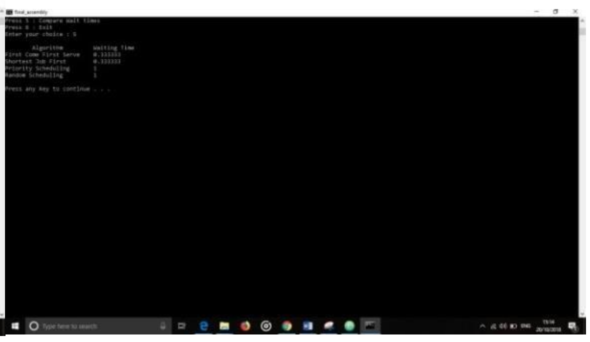
```
Welcome to Smart Scheduling System

Enter the number of trips you want to schedule : 3

Trip1

Enter the Booking Date (dd-mm-year) : 1-1-2018
Enter the Travel time of your trip : 1


Trip2

Enter the Booking Date (dd-mm-year) : 2-1-2018
Enter the Travel time of your trip : 2


Trip3

Enter the Booking Date (dd-mm-year) : 3-1-2018
Enter the Travel time of your trip : 3


***********************************************

SCHEDULE TYPE
Press 1 : First come first Serve
Press 2 : Shortest Job first
Press 3 : Priority
Press 4 : Random
Press 5 : Compare Wait times
Press 6 : Exit
Enter your choice : 1

First Come First Serve

Trip  Trip Time   Start Date  Completion Date  Waiting Time
 1        1        1:1:2018       2:1:2018          0
 2        2        2:1:2018       4:1:2018          0
 3        3        4:1:2018       7:1:2018          1

Average waiting time = 0.333333

***********************************************
```



```
SCHEDULE TYPE
Press 1 : First come first Serve
Press 2 : Shortest Job first
Press 3 : Priority
Press 4 : Random
Press 5 : Compare Wait times
Press 6 : Exit
Enter your choice : 2

 Shortest Job First


Trip  Trip Time   Start Date  Completion Date  Waiting Time
 1        1        1:1:2018       2:1:2018          0
 2        2        2:1:2018       3:1:2018          0
 3        3        4:1:2018       5:1:2018          1

Average waiting time = 0.333333


***********************************************

SCHEDULE TYPE
Press 1 : First come first Serve
Press 2 : Shortest Job first
Press 3 : Priority
Press 4 : Random
Press 5 : Compare Wait times
Press 6 : Exit
Enter your choice : 3

Priority Scheduling

Enter the Priority of Process 1 : 4
Enter the Priority of Process 2 : 2
Enter the Priority of Process 3 : 1
Trip  Trip Time  Priority Start Date  Completion Date  Waiting Time
 1        1          4      1:1:2018     2:1:2018          0

 2        2          2      2:1:2018     5:1:2018          1

 3        3          1      5:1:2018     8:1:2018          2

Average waiting time is : 1
```

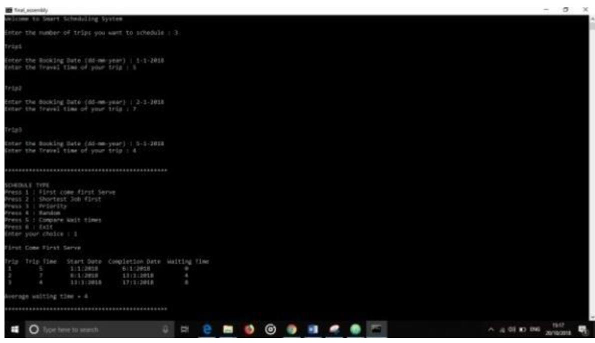


```
***********************************************

SCHEDULE TYPE
Press 1 : First come first Serve
Press 2 : Shortest Job first
Press 3 : Priority
Press 4 : Random
Press 5 : Compare Wait times
Press 6 : Exit
Enter your choice : 4

Random Scheduling

Random entry of Priorities

 Random entry of Priorities

 Random entry of Priorities

 Trip  Trip Time  Priority Start Date  Completion Date  Waiting Time
 1        1          3      1:1:2018     2:1:2018          0

 2        2          1      2:1:2018     5:1:2018          1

 3        3          2      5:1:2018     8:1:2018          2

Average waiting time is : 1

***********************************************

SCHEDULE TYPE
Press 1 : First come first Serve
Press 2 : Shortest Job first
Press 3 : Priority
Press 4 : Random
Press 5 : Compare Wait times
Press 6 : Exit
Enter your choice : 5

        Algorithm        Waiting Time
First Come First Serve    0.333333
Shortest Job First        0.333333
Priority Scheduling       1
```

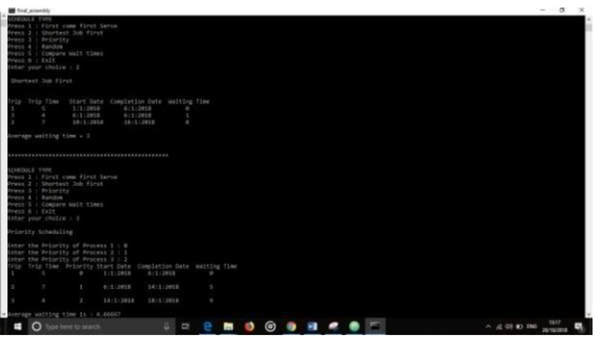


```
Press 5 : Compare Wait times
Press 6 : Exit
Enter your choice : 5

        Algorithm        Waiting Time
First Come First Serve    0.333333
Shortest Job First        0.333333
Priority Scheduling       1
Random Scheduling         1

Press any key to continue . . .
```

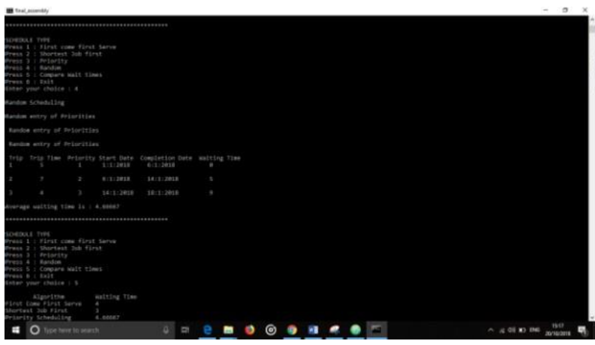
SET 2



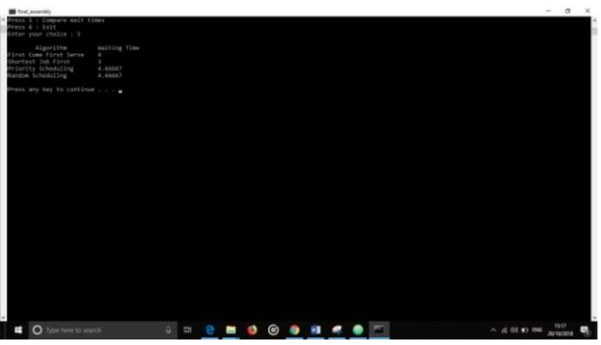
```
Welcome to Smart Scheduling System

Enter the number of trips you want to schedule : 3

Trip1

Enter the Booking Date (dd-mm-year) : 1-1-2018
Enter the Travel time of your trip : 5


Trip2

Enter the Booking Date (dd-mm-year) : 2-1-2018
Enter the Travel time of your trip : 7


Trip3

Enter the Booking Date (dd-mm-year) : 5-1-2018
Enter the Travel time of your trip : 4


***********************************************

SCHEDULE TYPE
Press 1 : First come first Serve
Press 2 : Shortest Job first
Press 3 : Priority
Press 4 : Random
Press 5 : Compare Wait times
Press 6 : Exit
Enter your choice : 1

First Come First Serve

Trip  Trip Time   Start Date  Completion Date  Waiting Time
 1        5        1:1:2018       6:1:2018          0
 2        7        6:1:2018       13:1:2018         4
 3        4        13:1:2018      17:1:2018         8

Average waiting time = 4

***********************************************
```



```
SCHEDULE TYPE
Press 1 : First come first Serve
Press 2 : Shortest Job first
Press 3 : Priority
Press 4 : Random
Press 5 : Compare Wait times
Press 6 : Exit
Enter your choice : 2

 Shortest Job First


Trip  Trip Time   Start Date  Completion Date  Waiting Time
 1        5        1:1:2018       6:1:2018          0
 3        4        6:1:2018       6:1:2018          1
 2        7        10:1:2018      18:1:2018         8

Average waiting time = 3


***********************************************

SCHEDULE TYPE
Press 1 : First come first Serve
Press 2 : Shortest Job first
Press 3 : Priority
Press 4 : Random
Press 5 : Compare Wait times
Press 6 : Exit
Enter your choice : 3

Priority Scheduling

Enter the Priority of Process 1 : 0
Enter the Priority of Process 2 : 1
Enter the Priority of Process 3 : 2
Trip  Trip Time  Priority Start Date  Completion Date  Waiting Time
 1        5          0      1:1:2018     6:1:2018          0

 2        7          1      6:1:2018     14:1:2018         5

 3        4          2      14:1:2018    18:1:2018         9

Average waiting time is : 4.66667
```

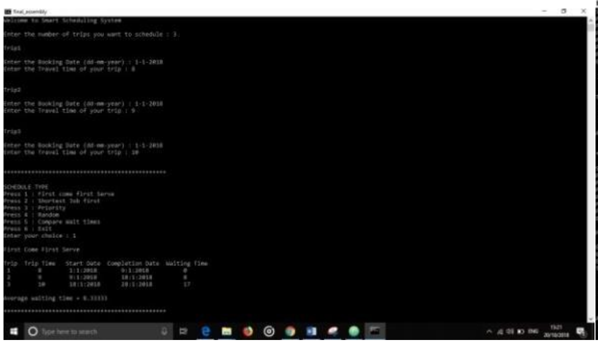


```
***********************************************

SCHEDULE TYPE
Press 1 : First come first Serve
Press 2 : Shortest Job first
Press 3 : Priority
Press 4 : Random
Press 5 : Compare Wait times
Press 6 : Exit
Enter your choice : 4

Random Scheduling

Random entry of Priorities

 Random entry of Priorities

 Random entry of Priorities

 Trip  Trip Time  Priority Start Date  Completion Date  Waiting Time
 1        5          1      1:1:2018     6:1:2018          0

 2        7          2      6:1:2018     14:1:2018         5

 3        4          3      14:1:2018    18:1:2018         9

Average waiting time is : 4.66667

***********************************************

SCHEDULE TYPE
Press 1 : First come first Serve
Press 2 : Shortest Job first
Press 3 : Priority
Press 4 : Random
Press 5 : Compare Wait times
Press 6 : Exit
Enter your choice : 5

        Algorithm        Waiting Time
First Come First Serve    4
Shortest Job First        3
Priority Scheduling       4.66667
```

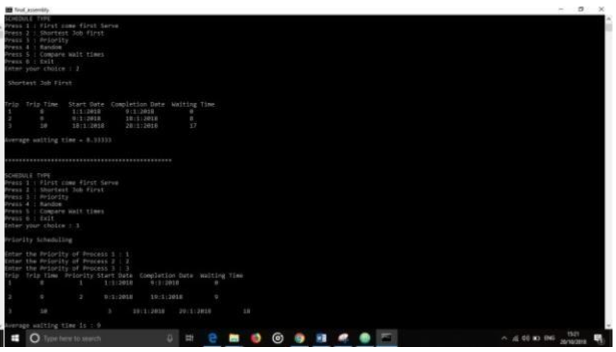


```
Press 5 : Compare Wait times
Press 6 : Exit
Enter your choice : 5

        Algorithm        Waiting Time
First Come First Serve    4
Shortest Job First        3
Priority Scheduling       4.66667
Random Scheduling         4.66667

Press any key to continue . . .
```

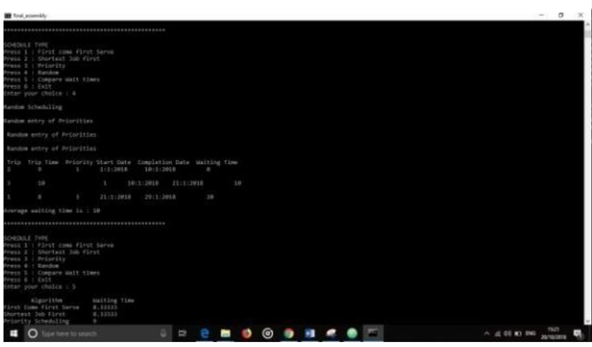
SET 3



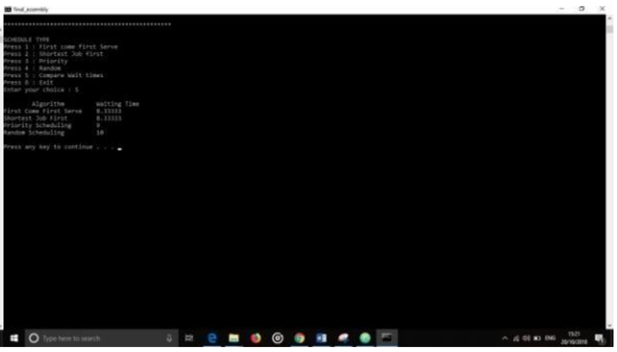
```
Welcome to Smart Scheduling System

Enter the number of trips you want to schedule : 3

Trip1

Enter the Booking Date (dd-mm-year) : 1-1-2018
Enter the Travel time of your trip : 8


Trip2

Enter the Booking Date (dd-mm-year) : 1-1-2018
Enter the Travel time of your trip : 9


Trip3

Enter the Booking Date (dd-mm-year) : 1-1-2018
Enter the Travel time of your trip : 10


***********************************************

SCHEDULE TYPE
Press 1 : First come first Serve
Press 2 : Shortest Job first
Press 3 : Priority
Press 4 : Random
Press 5 : Compare wait times
Press 6 : Exit
Enter your choice : 1

First Come First Serve

Trip  Trip Time   Start Date  Completion Date  Waiting Time
 1        8        1:1:2018       9:1:2018          0
 2        9        9:1:2018       18:1:2018         8
 3        10       18:1:2018      28:1:2018         17

Average waiting time = 8.33333

***********************************************
```



```
SCHEDULE TYPE
Press 1 : First come first Serve
Press 2 : Shortest Job first
Press 3 : Priority
Press 4 : Random
Press 5 : Compare wait times
Press 6 : Exit
Enter your choice : 2

 Shortest Job First


Trip  Trip Time   Start Date  Completion Date  Waiting Time
 1        8        1:1:2018       9:1:2018          0
 2        9        9:1:2018       18:1:2018         8
 3        10       18:1:2018      28:1:2018         17

Average waiting time = 8.33333


***********************************************

SCHEDULE TYPE
Press 1 : First come first Serve
Press 2 : Shortest Job first
Press 3 : Priority
Press 4 : Random
Press 5 : Compare wait times
Press 6 : Exit
Enter your choice : 3

Priority Scheduling

Enter the Priority of Process 1 : 1
Enter the Priority of Process 2 : 2
Enter the Priority of Process 3 : 3
Trip  Trip Time  Priority Start Date  Completion Date  Waiting Time
 1        8          1      1:1:2018     9:1:2018           0

 2        9          2      9:1:2018     19:1:2018          9

 3        10                  3      19:1:2018    29:1:2018        18

Average waiting time is : 9
```



```
***********************************************

SCHEDULE TYPE
Press 1 : First come first Serve
Press 2 : Shortest Job first
Press 3 : Priority
Press 4 : Random
Press 5 : Compare wait times
Press 6 : Exit
Enter your choice : 4

Random Scheduling

Random entry of Priorities

 Random entry of Priorities

 Random entry of Priorities

 Trip  Trip Time  Priority Start Date  Completion Date  Waiting Time
 2        9          1      1:1:2018     10:1:2018          0

 3        10                 1      10:1:2018    21:1:2018         10

 1        8          1      21:1:2018    29:1:2018          20

Average waiting time is : 10

***********************************************

SCHEDULE TYPE
Press 1 : First come first Serve
Press 2 : Shortest Job first
Press 3 : Priority
Press 4 : Random
Press 5 : Compare wait times
Press 6 : Exit
Enter your choice : 5

       Algorithm        Waiting Time
First Come First Serve   8.33333
Shortest Job First       8.33333
Priority Scheduling      9
```



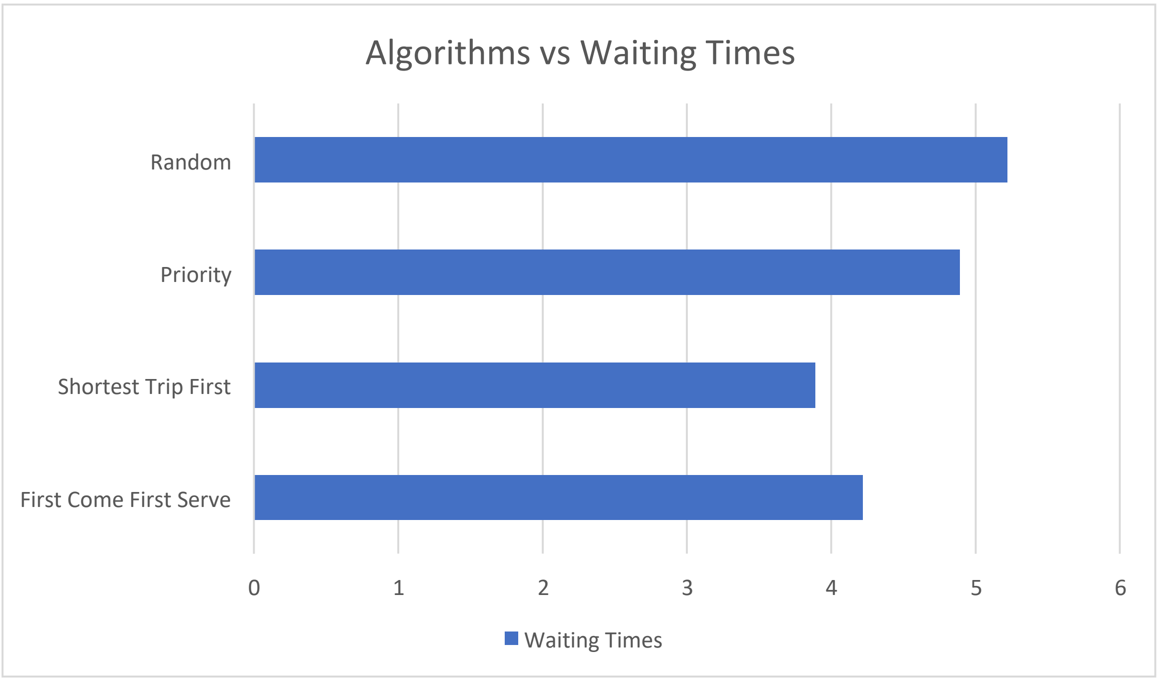
```
***********************************************

SCHEDULE TYPE
Press 1 : First come first Serve
Press 2 : Shortest Job first
Press 3 : Priority
Press 4 : Random
Press 5 : Compare wait times
Press 6 : Exit
Enter your choice : 5

       Algorithm        Waiting Time
First Come First Serve   8.33333
Shortest Job First       8.33333
Priority Scheduling      9
Random Scheduling        10

Press any key to continue . . .
```

Our fundamental idea is to lessen the holding up time of the client with the goal that their travel times are distributed in light of their picked time and the ever-increasing number of clients can be effortlessly overseen by the travel organization.

Observations

Algorithm	**Waiting Time**
First Come First Serve	**4.22**
Shortest Trip First	**3.89**
Priority	**4.89**
Random (varies with the order)	**5.22**

Graph



Conclusions

1) FCFS can cause long waiting times, especially when the first job takes too much CPU time.

2) Shortest Job First Algorithm can cause starvation. Consider a situation when long list of trips is there in ready queue and shorter processes keep coming.

3) SJF is optimal in terms of average waiting time for a given set of trips to be scheduled,i.e., average waiting time is minimum with this scheduling, but problems is, how to know/predict time of next job.

We can see that Shortest first is the better Algorithm as compared to the rest based on equal priorities to everyone. The above graph may change its shape based on the input values but the main observation remains common in all the graphs. The Random Algorithm can sometimes fare better than Fist-Cum-First serve and Priority Scheduling as any random trip can be scheduled at any time.

References

ChitraNayal. (n.d.). Dijkstra's shortest path algorithm. Retrieved from https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/

Chow, J. (n.d.). The Traveling Salesman Problem. Retrieved from file:///C:/Users/Dell/Desktop/ch11.pdf

KPS. (n.d.). Knowledge Management within the Travel and Tourism Industry. Retrieved from file:///C:/Users/Dell/AppData/Local/Packages/Microsoft.MicrosoftEdge_8wekyb3d8bbwe/TempState/Downloads/Whitepaper-Knowledge-Management-in-Travel-and-Tourism (1).pdf

Shi, S. M. ; H. Y. ; M. (n.d.). Developing a Creative Travel Management System Based on Software Reuse and Abstraction Techniques. Retrieved from https://ieeexplore.ieee.org/document/8029967

Shiyu Wang, L. C. (n.d.). Application of Travel Management System Based on Route Inquiry. Retrieved from http://www.sersc.org/journals/IJSH/vol9_no6_2015/15.pdf

Y.A. Adekunle1, Z.O. Ogunwobi2, A. Sarumi Jerry3, B.T. Efuwape2, Seun Ebiesuwa1, and J.-P. A. (n.d.). A Comparative Study of Scheduling Algorithms for Multiprogramming in Real-Time Systems. Retrieved from https://pdfs.semanticscholar.org/8d94/009fbdfe5ab2180c133d7b4806146cfdff92.pdf

Yan, M. (n.d.). *No Title*. Retrieved from http://math.mit.edu/~rothvoss/18.304.3PM/Presentations/1-Melissa.pdf