WWJMRD 2017; 3(12): 131-138
www.wwjmrd.com
International Journal
Peer Reviewed Journal
Refereed Journal
Indexed Journal
UGC Approved Journal
Impact Factor MJIF: 4.25
e-ISSN: 2454-6615

Shreyas Gonjari
School of Computer Science and Engineering, VIT, Vellore, Tamil Nadu, India

Gayathri. P
School of Computer Science and Engineering, VIT, Vellore, Tamil Nadu, India

Gaurav Gupta
School of Computer Science and Engineering, VIT, Vellore, Tamil Nadu, India

Santhi. H
School of Computer Science and Engineering, VIT, Vellore, Tamil Nadu, India

Shaik Naseera
School of Computer Science and Engineering, VIT, Vellore, Tamil Nadu, India

Ashruf Ahmed Siddiqui
School of Computer Science and Engineering, VIT, Vellore, Tamil Nadu, India

Gopichand. G
School of Computer Science and Engineering, VIT, Vellore, Tamil Nadu, India

Geraldine Bessie Amali
School of Computer Science and Engineering, VIT, Vellore, Tamil Nadu, India

Correspondence:
Shreyas Gonjari
School of Computer Science and Engineering, VIT, Vellore, Tamil Nadu, India

Music Playlist Manager Using Fisher-Yates Shuffling Algorithm and Sorting

Gayathri. P, Gaurav Gupta, Santhi. H, Shaik Naseera, Ashruf Ahmed Siddiqui, Gopichand. G, Geraldine Bessie Amali, Shreyas Gonjari

Abstract
In today's world music plays an important role in our lives. Music touches a person's soul and aids him to sympathetically exhibit humanity and unspoken desire within himself. Music helps bringing people together. It provides a platform where people experience similar emotions. There is nothing other than music to relieve one's soul and uplift it. Music provides us with the opportunity to express our feeling be it joy, passion, love, anger and other kind of feelings. Thus having our favorite music hands on is of high importance in today's world. The solution for it is having a Music playlist. Further listing to the same music repeatedly on the playlist can become boring and thus shuffling the music around in the music playlist is also important. Shuffling can be achieved through various techniques. The Fisher-Yates Algorithm (Knuth Algorithm) and sorting are two leading methods for achieving random shuffling. So having a comparison of the two shuffling techniques is a necessity. In this paper, based on various factors of comparison the Fisher-Yates Shuffle and the Sorting techniques for random shuffling have been compared. And based on the results of comparison we have come up with the more optimal of the two shuffling techniques.

Keywords: Music, Music Playlist, Shuffling, Fisher-Yates Algorithm, Knuth Algorithm, Sorting techniques Random Permutation, Domain-Specific Data Structures

Introduction
Music is any pleasant sound either vocal or instrumental that makes us experience joy and higher harmony. Music plays an essential role in human life. Music is believed to have relaxing and healing powers as well. Music helps us to relax and has a healing and soothing power. The melodious notes of music are believed to have the ability to heal the physical or mental fatigue of a tired and weary person. The power of music can draw people from all corners of the world and uplift their spirits as well as emotions. Further the best or optimal way to express one's feeling be it love, anger, joy, sorrow, longing and other feelings are through music. Thus music is an important aspect of human beings and is an important and vital pillar in their lives. Now having our favorite music hands on is extremely necessary and one solution to this problem is having or maintaining a music playlist. As per the approach of [1] there are different ways to create playlists. One extreme is to very carefully select each piece and the order in which the pieces are played. Another extreme is to randomly shuffle all pieces in a collection. While the first approach is very time consuming, the second approach produces useless results if the collection is very diverse in [1].. Knight and Rickard uncovered that unwinding music lessened circulatory strain and heart rate after an upsetting undertaking; also, the level of subjective tension was decreased after the introduction of unwinding music [1]. Stratton and Zalanowski [3] led investigations and found that inclination, recognition or past encounters with the music overriding affect positive conduct change than different sorts of music [2]. Listening to the same music repeatedly can be extremely boring and irritating. Thus shuffling of songs in the playlist is an important task and should be given considerable weight age. Shuffling is a procedure wherein the songs are randomly ordered without any preference to a song or following any particular order of placing the elements, in this case songs present in the music playlist. The task of random

shuffling can be achieved through various shuffling techniques. Few shuffling techniques include shuffling algorithms and using sorting to randomize elements. There are various shuffling algorithms available in today's world. A few of the shuffling algorithms are Faro's Shuffle, the Fisher-Yates Shuffle (Knuth shuffle) Algorithm and so on. Amongst all the shuffling algorithms, the Fisher-Yates Shuffle (Knuth shuffle) Algorithm is the most optimal and widely used algorithm. A lot of research has been done and is still going in order to improve the Fisher-Yates Shuffle Algorithm and it has been applied to various real life applications. Another famous shuffling technique is using sorting to attain the task of random shuffling of elements. Sorting has been widely used to achieve shuffling and various sorting techniques have been tried and tested in order to come up with the most optimal sorting technique. A lot of research has been done in this field in order to come up with the best sorting technique to achieve random shuffling. Thus shuffling of songs to induce change in life and to overcome boredom in life is extremely important. Reports and research suggests that listening to the same music repeatedly can cause a person to go into depression and may push him to hate music and to give on music itself. As we know that change is extremely important in life and is beneficial as well. Thus change of the music being heard by a person is equally necessary.

Rest of the paper is organized as follows: Section II describes about the existing works in the literature. Section III contains the proposed methodology. Section IV illustrates the results obtained and Section V contains the conclusion.

Literature Review

We have discussed below the literature review on the topic. An epitome relates for the most part to a technique for playback in a media player. The strategy incorporates recovering no less than one tune from a media library in an irregular mode on a media player and playing the no less than one tune through the media player. The technique additionally incorporates starting a collection mode on the media player and recovering outstanding melodies on a collection related with the no less than one tune.

A technique including: getting a determination of a first arrangement of media documents for play in an arbitrary mode; recovering a sound document of the primary arrangement of media records from a media library on a media player, the sound document related with a collection; beginning play of the sound record through the media player; starting a collection mode on the media player; recovering an extra sound record related with the collection with which the sound document is related; adding the extra sound document to the primary arrangement of media records; beginning, by a processor, play of the extra sound document after a fruition of play of the sound record; and consequently giving back the media player to the arbitrary mode for play of the principal set of media records after a fulfillment of play of the extra sound document. The strategy for claim 1, wherein a majority of extra sound records are recovered and requested for playback after fulfillment of play of the sound document.

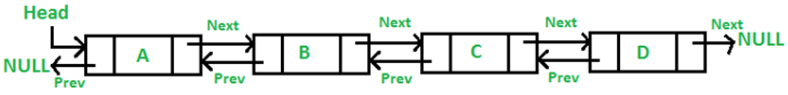
This creation relates by and large to media players, all the more especially, to frameworks and techniques for collection playback in an irregular mode [10]on a media player. The media player application can be additionally designed to playback media records in a few modes. One mode can be a consecutive mode where media records in the media library or a playlist are played back in successive request in view of alpha-numeric, time-dated or other client gave criteria. Another mode can be an irregular or rearrange mode. The arbitrary mode can play back media document inside the media library or a playlist in an irregular request.

Proposed Methodlogy

There are various methods in which a music playlist can be created. One can use n number of data structures to implement a music playlist. In this project we have implemented a music playlist using doubly linked list where in the data element of the each node in the linked list contains the song name, the link to the next node as well as the previous node is also contained in each node so that one can access the previous as well as the next song. The user has the options to add and delete a song at the beginning, end and at any random location with respect to the music playlist. Further the program also display's the current song which is being played. The user can sort the songs in the music playlist. User can also search for any song and at the same time update any song already present in the music playlist. The user is also permitted to shuffle the songs in the music playlist using two methods, firstly using the Fisher-Yates Shuffle algorithm and secondly using sorting in order to achieve random shuffling.

Doubly Linked List

A **D**oubly **L**inked **L**ist (DLL) (Linked List Introduction [1]) contains an extra pointer, typically called *previous pointer*, together with next pointer and data which are there in singly linked list.



Following are advantages/disadvantages of doubly linked list over singly linked list -

Advantages

1) A DLL can be traversed in both forward and backward direction.
2) The erase operation in DLL is more effective if pointer to the hub to be erased is given. In independently connected rundown, to erase a hub, pointer to the past hub is required. To get this past hub, here and there the rundown is crossed. In DLL, we can get the past hub utilizing past pointer.

Disadvantages

1) Every node of DLL Require extra space for a previous

pointer. It is possible to implement DLL with single pointer.

2) All operations require an extra pointer previous to be maintained. For example, in insertion, we need to modify previous pointers together with next pointers. For example in following functions for insertions at different positions, we need 1 or 2 extra steps to set previous pointer.

Figure 1 depicts the flow of proposed work.

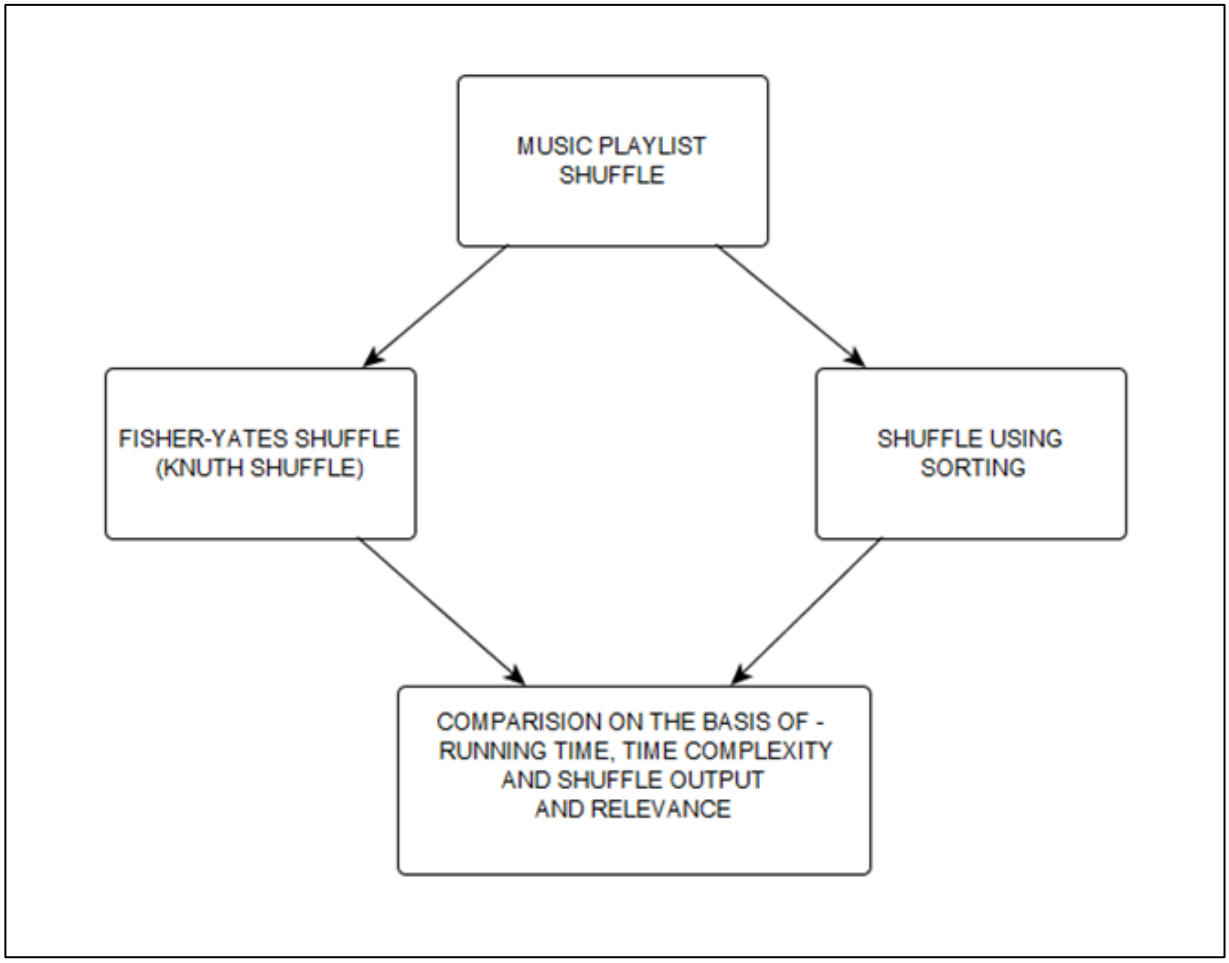


Fig.1: Flow of proposed work

Fisher-Yates Algorithm Approach

The Fisher-Yates Shuffling algorithm is primarily used to obtain a random permutation of a fixed sequence, in simpler terms the algorithm randomly shuffles the sequence. The basic functionality of the algorithm can be explained as – it puts all the elements of the sequence into a box and then continuously identifies the subsequent element by randomly drawing an element from the box until no element remains. Its approach is similar to that of a lucky draw wherein nobody can predict what the next element will be. The algorithm generates an unbiased permutation: wherein every permutation has the same possibility to occur. The Fisher-Yates Algorithm also known as the Knuth Algorithm has obtained its name after its founders namely Frank Yates and Ronald Fisher. The algorithm was first described in 1983 by Yates and Fisher in their Statistical table book for agriculture, biological and medical research. The Fisher-Yates Shuffle algorithm designed for computer science purpose was first described by Richard Durstenfeld in 1964 and was made famous by Donald Knuth in his book "The Art of Computer Programming" as 'algorithm P'.

The Fisher-Yates Shuffle Algorithm for computer use is as described in figure 2.

```
-- To shuffle an array a of n elements (indices 0..n-1):
for i from n−1 downto 1 do
     j ← random integer such that 0 ≤ j ≤ i
     exchange a[j] and a[i]
```

Fig.2: Fisher-Yates Algorithm [2]

Another equivalent version of the Fisher-Yates Shuffle Algorithm which shuffles the array from the lowest index to the highest index is shown in figure 3

```
-- To shuffle an array a of n elements (indices 0..n-1):
for i from 0 to n-2 do
     j ← random integer such that i ≤ j < n
     exchange a[i] and a[j]
```

Fig.3: Fisher-Yates Shuffle Algorithm (From Lower to Higher Index) [2]

In this paper we have kept the basic functionality for both the codes of the two approaches the same wherein as mentioned before the user can add, delete, sort, update, play next or previous song in the playlist. The only functionality where the codes change is the shuffling part where both the approaches have been applied in the respect codes.

In the first code where the Fisher-Yates Shuffle Algorithm is implemented, the song from each node of the doubly linked list is taken and an array is created with the song as the array element. Now using the indices of the array we have implemented the Fisher-Yates Shuffle algorithm. The Randomize function in C programming language is used to generate any random number within the range 0 – N, where N is the size of the array. Now the song at this index is swapped with the song at the last location as per the Fisher-Yates Shuffle algorithm. And this process is repeated until all the indices of the array have been generated by reducing the value of N by 1 after each swap until all the elements have been swapped from their original location. Then array thus obtained from the above procedure is then converted back to a doubly linked list. And this the manner in which the Fisher-Yates Shuffle Algorithm has been implemented in order to attain random shuffling of songs in the music playlist.

Shuffling Using Sorting

The sorting technique has also been used in this project to implement random sorting. Firstly the song from each node is taken and a 2-D array of Nx2, where N is the number of songs in the playlist, has been formed out of it. Now any random number is assigned to the array elements i.e. the songs using the Randomize function. Then the array is sorted using the bubble sort technique in the ascending order with respect to the random numbers assigned to the array. The array thus obtained from the above steps is then converted back to a doubly linked list. This is the manner in which sorting has been used in order to obtain a random permutation i.e. shuffled sequence of the songs present in the playlist.

Bubble Sort Algorithm

Bubble sort calculation [4] begins by looking at the initial two components of an exhibit and swapping if essential, i.e., on the off chance that you need to sort the components of cluster in rising request and if the primary component is more noteworthy than second then, you have to swap the components at the same time, if the main component is littler than second, you mustn't swap the component. At that point, again second and third components are looked at and swapped in the event that it is essential and this procedure go ahead until last and second last component is analyzed and swapped. This finishes the initial step of air pocket sort. On the off chance that there are n components to be sorted then, the procedure specified above ought to be rehashed n-1 times to get required outcome. Yet, for better execution, in second step, last and second last components are not thought about because; the correct component is naturally put finally after initial step. Correspondingly, in third step, last and second last and second last and third last components are not looked at et cetera. A figure 4 and figure 5 show the working of bubble sort algorithm and pseudocode respectively [16].

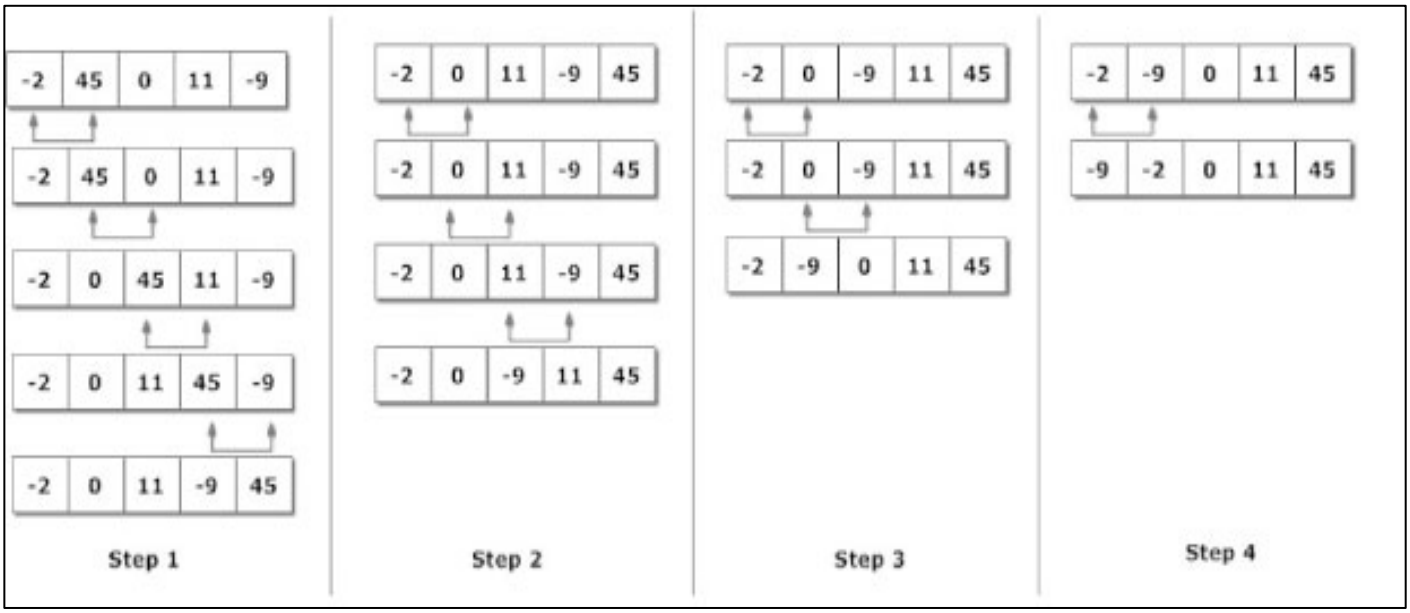


Fig.4: Working of bubble sort algorithm

```
procedure bubbleSort( A : list of sortable items )
    n = length(A)
    repeat
        swapped = false
        for i = 1 to n-1 inclusive do
            /* if this pair is out of order */
            if A[i-1] > A[i] then
                /* swap them and remember something changed */
                swap( A[i-1], A[i] )
                swapped = true
            end if
        end for
    until not swapped
end procedure
```

Fig.5: Pseudo Code for Bubble Sort

Results and Discussion

Fisher–Yates shuffle Algorithm works in O (n) time complexity [12]. The assumption here is, we are given a: function rand() that generates random number in O(1) time. The idea is that we start from the last element, swap it with a randomly selected element from the whole playlist (including last). Now consider the playlist index from 0 to n-2 (size reducing by 1), and repeated the steps till we hit the first element. The Fisher–Yates rearrange is unprejudiced, so that each change is similarly likely. The present day variant of the calculation is likewise rather productive, requiring just time corresponding to the quantity of things being rearranged and no extra storage room. Fisher–Yates rearranging [5] is like arbitrarily picking numbered tickets (combinatory: recognizable items) out of a cap without substitution until there are none cleared out. Advantages – 1) Asymptotic time and space intricacy are ideal. O(n) time and O(1) space. 2) Ensured to deliver impartial outcomes.

Combining two arrangements by over and over picking one of them with equivalent likelihood (until the decision is constrained by the fatigue of one succession) does not deliver comes about with a uniform dissemination; rather the likelihood to pick a grouping ought to be corresponding to the quantity of components left in it. Truth be told no strategy that utilizations just two-way irregular occasions with equivalent likelihood ("coin flipping"), rehashed a limited number of times, can create changes of a succession (of more than two components) with a uniform dissemination [8], in light of the fact that each execution way will have as likelihood a judicious number with as denominator an energy of 2, while the required likelihood 1/n!. For every conceivable change is not of that frame. Since we know that bubble sort has time complexity $O(n^2)$ while Fisher-Yates has time complexity O(n), thus the Code using the Fisher-Yates Algorithm runs with a better time constrain.

Figure 6 and figure 7 very clearly depicts that the Fisher-Yates algorithm does better in the compilation (Execution time) time.

PLAY_FYA_SHUFFLE.cpp | PLAY_SORT_SHUFFLE.cpp

```
1   #include <stdio.h>
2   #include <stdlib.h>
3   #include<string.h>
4   #include<math.h>
5   #include<time.h>
6   struct node
7   {
8       struct node *prev;
9       char n[30];
10      struct node *next;
11  }*front,*temp,*rear,*temp2,*temp4,*ptr;
12
13  void insert1();
14  void insert2();
15  void insert3();
16  void traversebeg();
17  void sort();
18  void search();
19  void update();
20  void delete1();
21  void shuffle();
22  static int rand_int();
23
24  void go_next()
25  {
26      if(ptr==rear)
```

Compile Log | Debug | Find Results | Close

```
Compilation results...
--------
- Errors: 0
- Warnings: 0
- Output Filename: C:\Users\Admin\Desktop\PLAY_FYA_SHUFFLE.exe
- Output Size: 134.9775390625 KiB
- Compilation Time: 0.47s
```

Fig.6: Fisher-Yates Compilation Time

PLAY_FYA_SHUFFLE.cpp | PLAY_SORT_SHUFFLE.cpp

```
#include <stdio.h>
#include <stdlib.h>
#include<string.h>
#include<math.h>
#include<time.h>
struct node
{
    struct node *prev;
    char n[30];
    struct node *next;
}*front,*temp,*rear,*temp2,*temp4,*ptr;

void insert1();
void insert2();
void insert3();
void traversebeg();
void sort();
void search();
void update();
void delete1();
void shuffle();
static int rand_int(int n);

void go_next()
{
    if(ptr==rear)
```

Compile Log | Debug | Find Results | Close

```
Compilation results...
--------
- Errors: 0
- Warnings: 0
- Output Filename: C:\Users\Admin\Desktop\PLAY_FYA_SHUFFLE.exe
- Output Size: 134.9775390625 KiB
- Compilation Time: 1.97s
```

Fig.7: Shuffling using Sorting Compilation Time

Complexity of the code has also been considered for the basics of comparison. The code for the Fisher-Yates Shuffle Algorithm is smaller and easier to implement when compared to the code for the shuffling using sorting. Another factor based on which we have compared is the shuffled array obtained from both the shuffling techniques. When this factor is considered, both the shuffling techniques fair in a similar fashion and it can be considered that both the shuffling techniques generate random permutations each time the code is executed.

Figure 8 and figure 9 depict the Shuffle generated from the Fisher-Yates Shuffle Algorithm and sorting technique.

C:\Users\Admin\Desktop\PLAY_FYA_SHUFFLE.exe

```
Playlist Songs from begining :
 F
E
D
A
Z
B
C
G
H
I
1 - Insert Song at beginning
2 - Insert Song at end
3 - Insert Song at position i
4 - Delete Song at i
5 - Display Songs in Playlist
6 - Search for Song
7 - Sort the Playlist
8 - Update a Song
9 - Next song
10 - Previous song
11 - Shuffle
12 - Exit
 ----------------------------------------
C
 ----------------------------------------

 Enter choice : 11

 1 - Insert Song at beginning
 2 - Insert Song at end
 3 - Insert Song at position i
 4 - Delete Song at i
 5 - Display Songs in Playlist
 6 - Search for Song
 7 - Sort the Playlist
 8 - Update a Song
 9 - Next song
 10 - Previous song
 11 - Shuffle
 12 - Exit
 ----------------------------------------
Z
 ----------------------------------------

 Enter choice : 5

 Playlist Songs from begining :
  F
 A
 D
 E
 C
 G
 Z
 B
 H
```

Fig.8: Shuffle generated from the Fisher-Yates Shuffle Algorithm

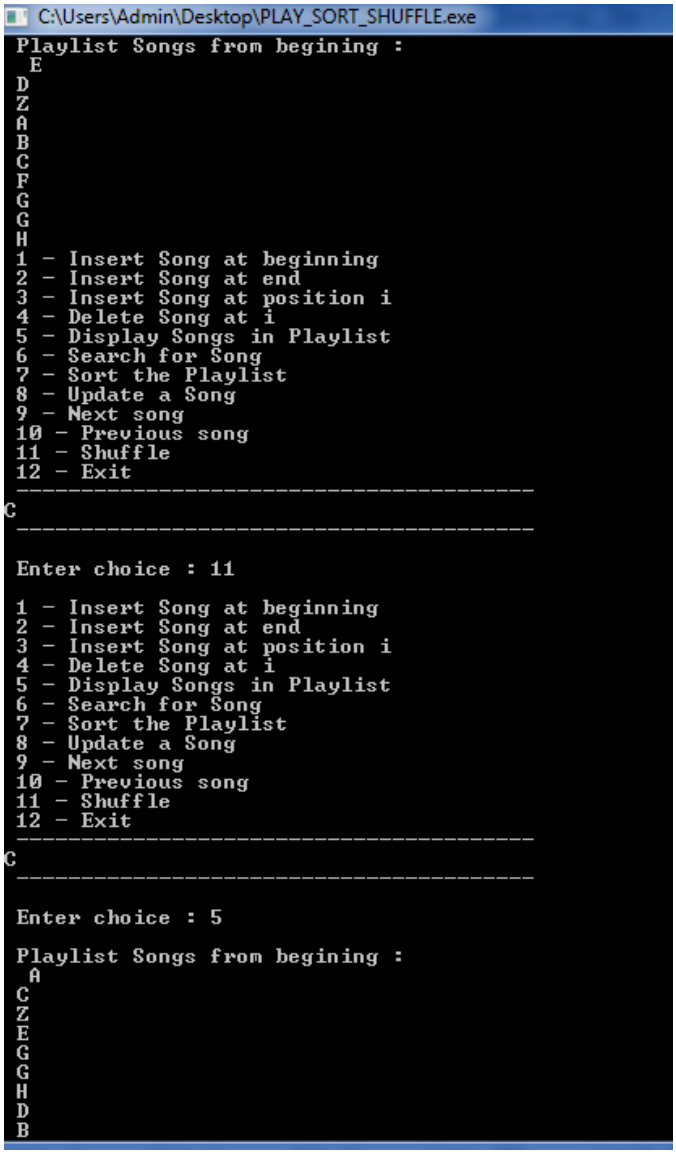


Fig.9: Shuffle generated using the Sorting Technique

One more possible way for sorting the songs could have been on the basis of genre as in the music field there are so many genres like jazz, Blues, classic and sorting according to the type of genres can be a good addition. As now a day's people are very particular or choosey of their music choices and in this world of sophistication everything needs to be precisely placed and well arranged.

Conclusion

Since music is such a major some portion of our lives, we thought it would help to observe a portion of the ways we respond to it without figuring it out. "Without music, life would be a mistake" –Friedrich Nietzsche. Similar research shows this correlation for exercise and motor skills in the same way, which is also fascinating. Based on the results, we conclude that the Fisher-Yates Shuffle algorithm is a better technique to implement shuffling and generate a random permutation. Both the techniques have similar random permutations generated however when it comes to time complexity and code compatibility, the Fisher-Yates Shuffle algorithm just edges the shuffling techniques using Sorting. The work can be further improved by also considering various sorting techniques and comparing the Fisher-Yates Shuffle algorithm to all the other sorting techniques. Also repeated or same entries of songs can also be considered and the random permutation thus obtained from both the shuffling techniques can also be compared.

References

1. Fisher, Ronald A.; Yates, Frank (1948) [1938]. Statistical tables for biological, agricultural and medical research (3rd ed.). London: Oliver & Boyd.
2. Black, Paul E. (2005-12-19). "Fisher–Yates shuffle". Dictionary of Algorithms and Data Structures
3. Ade-ibijola, Abejide Olu. International Journal of Computer Applications; New York 54.11 (2012).
4. Harriman, Martin, "Sampling Without Replacement: Durstenfeld-Fisher-Yates Permutation for Very Large Permutation Sizes", Technical Disclosure Commons, (March 20, 2017) http://www.tdcommons.org/dpubs_series/430
5. Durstenfeld, Richard, "Algorithm 235: Random Permutation," Communications of the ACM, volume 7 number 7.
6. Knuth, Donald E., The Art of Computer Programming, Volume 2: Seminumerical Algorithms, third edition, Boston, 1997.
7. Algorithm" (PDF). In F. Chyzak (ed.). INRIA Research Report. Algorithms Seminar 2002–2004 5542. Summary by Éric Fusy.

8. "Provably perfect shuffle algorithms". Oleg Kiselyov. 3 Sep 2001. Retrieved 2013-07-09.
9. "A simple shuffle that proved not so simple after all". Require 'brain'. 2007-06-19. Retrieved 2007-08-09.
10. "Doing the Microsoft Shuffle: Algorithm Fail in Browser Ballot". Rob Weir: An Antic Disposition. 2010-02-27. Retrieved 2010-02-28.
11. "Writing a sort comparison function, redux". Require 'brain'. 2009-05-08. Retrieved 2009-05-08.
12. [Jeff Atwood (December 7, 2007). "The Danger of Naïveté". Coding Horror: programming and human factors
13. Knuth (1998). Seminumerical algorithms. The Art of Computer Programming 2 (3rd ed.). Boston: Addison–Wesley. pp. 145–146
14. [14.]"Introduction to Algorithms" by Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein
15. [15.] O Astrachan - ACM SIGCSE Bulletin, 2003 - dl.acm.org