c:

```
#include <stdio.h>
#include <stdlib.h>

#define ll long long
#define gc getchar_unlocked
#define SS 500
#define BS 300

int getn(){
  int n = 0, c = gc(), f = 1;
  while(c != '-' && (c < '0' || c > '9')) c = gc();
  if(c == '-') f = -1, c = gc();
  while(c >= '0' && c <= '9') n = (n<<3) + (n<<1) + c - '0', c = gc();
  return n * f;
}

ll absl(ll n){ return (n < 0) ? -n : n; }
int min(int a, int b){ return (a < b) ? a : b; }

int N, f,k, a[100000], r[10000][2],rn;
ll d;

void shift(int L, int R){
  int i, t;
  if(R-L+1 > k) return;
  for(i = L, t = a[L]; i < R; ++i) a[i] = a[i+1];
  a[R] = t, k -= R-L+1, f = 1;
  if(L < N/2 && R >= N/2) d += 2*a[N/2-1] - 2*a[R];
  r[rn][0] = L, r[rn][1] = R, ++rn;
}

int main(){
  int i,j;
  ll m,t;

  N = getn();
  for(i = 0; i < N; ++i) a[i] = getn();

  for(i = d = 0; i < N/2; ++i) d += a[i];
  for(i = N/2, m = 0; i < N; ++i) m += a[i];
  d -= m, k = N*2, rn = 0;
  while(1){
    if(!d) break;
    for(i = N/2-1, m = absl(d); i >= N/2-min(N/2, SS); --i){
      t = absl(d - 2*a[i] + 2*a[N/2]);
      if(t < m) m = t, j = i;
    }
    f = 0;
    if(m == absl(d)) shift(N/2, N/2+min(N/2-1, BS));
    else shift(j, N/2+min(N/2-1, BS));
    if(!f) break;
  }

  printf("%d\n", rn);
  for(i = 0; i < rn; ++i) printf("%d %d\n", r[i][0]+1, r[i][1]+1);
  return 0;
```

```
}
```

----

python:

```
from random import random
from _bisect import bisect_right

class ShufflerGenome(object):
    halfn = 0
    halfn1 = 0
    def __init__(self, a, lsum, rsum):
        self.array = a[:]
        self.shuffles = []
        self._left = 2*len(a)
        self._lsum = lsum
        self._rsum = rsum
        self._initWithRandomShuffles()

    def _initWithRandomShuffles(self):
        while True:
            if self._left < 2: return
            elif self._left < ShufflerGenome.halfn:
                u = ShufflerGenome.halfn - 1 - int(random()*(self._left>>1))
                v = ShufflerGenome.halfn + int(random()*(self._left>>1))
            else:
                u = int(random()*ShufflerGenome.halfn)
                v = ShufflerGenome.halfn + int(random()*ShufflerGenome.halfn1)
            if v - u + 1 > self._left: return
            self.shuffles.append((u, v))
            self._shuffle(u, v)

    def _shuffle(self, l, r):
        """ Shuffles current array in the range [l..r]
            Ex: Shuffling [1 2 3 4 5 6] in range [2..4]
                results in [1 3 4 2 5 6]"""
        self._left -= r - l + 1
        if self._left < 0: raise Exception
        self._lsum -= self.array[l]
        self._lsum += self.array[ShufflerGenome.halfn]
        self._rsum -= self.array[ShufflerGenome.halfn]
        self._rsum += self.array[l]
        p = self.array[l]
        for i in range(l, r):
            self.array[i] = self.array[i+1]
        self.array[r] = p

    def _unshuffle(self, l, r):
        """ Shuffles current array in the range [l..r]
            Ex: Shuffling [1 3 4 2 5 6] in range [2..4]
                results in [1 2 3 4 5 6]"""
        self._left += r - l + 1
        self._lsum += self.array[r]
        self._lsum -= self.array[ShufflerGenome.halfn1-1]
        self._rsum += self.array[ShufflerGenome.halfn1-1]
        self._rsum -= self.array[r]
        p = self.array[r]
        for i in range(r, l, -1):
            self.array[i] = self.array[i-1]
```

```
            self.array[l] = p

    def getCost(self):
        """ returns abs diff between sum of left and right halfs of array """
        return abs(self._lsum - self._rsum)

    def getFitness(self):
        """ returns total sum - (abs diff between sum of left and right halfs of array) """
        return self._lsum + self._rsum - self.getCost()

    def crossover(self, partner):
        t = min(len(self.shuffles), len(partner.shuffles))
        p = t >> 1
        # p = int(random()*t)
        for i in range(len(self.shuffles)-1, p, -1):
            self._unshuffle(*self.shuffles[i])
        self.shuffles = self.shuffles[:p+1]
        for i in range(p+1, len(partner.shuffles)):
            u, v = partner.shuffles[i]
            if v - u + 1 > self._left: return
            self._shuffle(u, v)

    def mutate(self, gene):
        u = int(random()*ShufflerGenome.halfn)
        v = ShufflerGenome.halfn + int(random()*ShufflerGenome.halfn1)
        pre = self.shuffles[gene][1] - self.shuffles[gene][0] + 1
        cur = v - u + 1
        if cur - pre > self._left: return
        for i in range(len(self.shuffles)-1, gene-1, -1):
            self._unshuffle(*self.shuffles[i])
        self.shuffles[gene] = (u, v)
        for i in range(gene, len(self.shuffles)):
            self._shuffle(*self.shuffles[i])

    def __str__(self):
        out = ""
        out += "[Cost: %d\tLeft:%d\tSwaps: %d] " % (self.getCost(), self._left, len(self.shuffles))
        for i in self.array: out += str(i) + " "
        return out

if __name__ == '__main__':
    n = int(input())
    a = [int(i) for i in input().split()]
    ShufflerGenome.halfn = n >> 1
    ShufflerGenome.halfn1 = (n + 1) >> 1
    lsum = sum(a[:ShufflerGenome.halfn])
    rsum = sum(a[ShufflerGenome.halfn:])
    minc = abs(lsum - rsum)
    minshufs = []

    population = 10
    mutationrate = 0.015
    candidates = [ShufflerGenome(a, lsum, rsum) for i in range(population)]

    for generation in range(50):
        cummulative_fitness = [0] * population
        cummulative_fitness[0] = candidates[0].getFitness()
        for i in range(1, population):
```

```
    cummulative_fitness[i] = cummulative_fitness[i-1] + candidates[i].getFitness()

# print("Generation %d:" % (generation))
# for i in candidates: print(i)
# print(cummulative_fitness)
# print()

p = bisect_right(cummulative_fitness, int(random() * cummulative_fitness[-1]))
q = bisect_right(cummulative_fitness, int(random() * cummulative_fitness[-1]))
c = 0
while p == q and c < n:
    q = bisect_right(cummulative_fitness, int(random() * cummulative_fitness[-1]))
    c += 1
if p != q: candidates[p].crossover(candidates[q])

for i in range(population):
    if candidates[i].getCost() < minc:
        minc = candidates[i].getCost()
        minshufs = candidates[i].shuffles[:]
    genes = len(candidates[i].shuffles)
    for gene in range(genes):
        p = random()
        if p < mutationrate:
            # mutating genes-1-gene so that the number of
            # unshufflings and shufflings can be reduced
            candidates[i].mutate(genes-1-gene)
            # Only one gene can mutate :P
            break
```

```
print(len(minshufs))
for i in minshufs:
    print(i[0]+1, i[1]+1)
```